BITS Pilani
Pilani Campus

Data Structures & Algorithms Design- SS ZG519
Lecture - 18

Dr. Padma Murali

Lecture 18 Topics

Complexity classes- P & NP

innovate achieve lead

BITS Pilani
Pilani Campus

- **Complexity classes (P and NP)**

P and NP

- Some problems are *intractable*:
  as they grow large, we are unable to solve them in reasonable time
- What constitutes reasonable time? Standard working definition: *polynomial time*
  - On an input of size n the worst-case running time is $O(n^k)$ for some constant k
  - Polynomial time: $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$
  - Not in polynomial time: $O(2^n)$, $O(n^n)$, $O(n!)$

P and NP

- Are some problems solvable in polynomial time?
  - Of course: every algorithm we've studied provides polynomial-time solution to some problem
  - We define **P** to be the class of problems solvable in polynomial time
- Are all problems solvable in polynomial time?
  - No: Turing's "Halting Problem" is not solvable by any computer, no matter how much time is given
  - Such problems are clearly intractable, not in **P**
  - A *Turing machine* is a mathematical model of a universal computer (any computation that needs polynomial time on a Turing machine can also be performed in polynomial time on any other machine)

P and NP

- **NP** (**N***ondeterministic* ***P****olynomial time)* is the set of all decision problems (question with yes-or-no answer) for which the 'yes'-answers can be **verified** in polynomial time (O(n^k) where n is the problem size, and k is a constant) by a deterministic Turing Machine. Polynomial time is sometimes used as the definition of *fast* or *quickly*.

- **P (P**olynomial time) is the set of all decision problems which can be **solved** in polynomial time by a deterministic Turing machine. Since it can solve in polynomial time, it can also be verified in polynomial time. Therefore P is a subset of NP.

P and NP

- P Decision problems for which there exists a poly-time algorithm.
- NP Decision problems for which there exists a poly-time certifier

- Hamiltonian-cycle problem is in **NP**:
  - Cannot solve in polynomial time
  - Easy to verify solution in polynomial time

NP-Complete

What is NP-Complete?

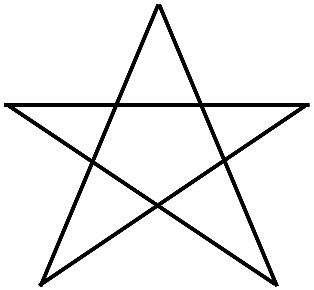
- A problem x that is in NP is also in NP-Complete if and only if every other problem in NP can be quickly (ie. in polynomial time) transformed into x. In other words:
- x is in NP, and
- Every problem in NP is reducible to x

An NP-Complete Problem: Hamiltonian Cycles

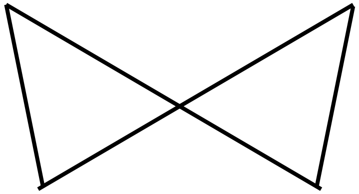
- An example of an NP-Complete problem:
  - A *hamiltonian cycle* of an undirected graph is a simple cycle that contains every vertex
  - The hamiltonian-cycle problem: given a graph G, does it have a hamiltonian cycle?

An NP-Complete Problem: Hamiltonian Cycles

- **Given:** a directed graph $G = (V, E)$, determine a simple cycle that contains each vertex in V
  - Each vertex can only be visited once
- **Certificate:**
  - Sequence: $\langle v_1, v_2, v_3, \ldots, v_{|V|} \rangle$
- **Verification:**
  1) $(v_i, v_{i+1}) \in E$ for $i = 1, \ldots, |V|$
  2) $(v_{|V|}, v_1) \in E$



hamiltonian



not hamiltonian

Reduction

- The crux of NP-Completeness is *reducibility*
  - Informally, a problem P can be reduced to another problem Q if *any* instance of P can be "easily rephrased" as an instance of Q, the solution to which provides a solution to the instance of P
    - This rephrasing is called *transformation*
  - Intuitively: If P reduces to Q, P is "no harder to solve" than Q

NP –COMPLETE PROBLEMS

- Given one NP-Complete problem, we can prove many interesting problems NP-Complete
  - Graph coloring (= register allocation)
  - Hamiltonian cycle
  - Hamiltonian path
  - Knapsack problem
  - Traveling salesman
  - Job scheduling with penalities

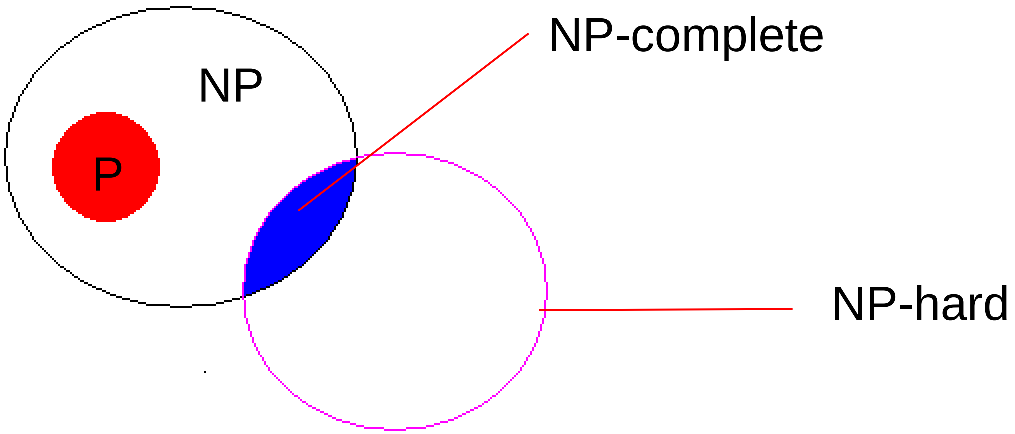
NP-Hard

- NP-Hard are problems that are at least as hard as the hardest problems in NP. Note that NP-Complete problems are also NP-hard. However not all NP-hard problems are NP (or even a decision problem), despite having 'NP' as a prefix. That is the NP in NP-hard does not mean 'non-deterministic polynomial time'. Yes this is confusing but its usage is entrenched and unlikely to change.
- A problem Y is NP-hard if X p Y for an NP-complete problem X.

NP-Hard and NP-Complete

- If P is *polynomial-time reducible* to Q, we denote this $P \leq_p Q$
- Definition of NP-Hard and NP-Complete:
  - If all problems $R \in$ **NP** are reducible to P, then P is *NP-Hard*
  - We say P is *NP-Complete* if P is NP-Hard and $P \in$ **NP**
- If $P \leq_p Q$ and P is NP-Complete, Q is also NP- Complete

P,NP,NPC and NP-hard



P & NP-Complete Problems

Shortest simple path

- Given a graph G = (V, E) find a **shortest** path from a source to all other vertices
- Polynomial solution: O(VE)

Longest simple path

- Given a graph G = (V, E) find a **longest** path from a source to all other vertices
- NP-complete

P & NP-Complete Problems

Euler tour

- G = (V, E) a connected, directed graph find a cycle that traverses each edge of G exactly once (may visit a vertex multiple times)
- Polynomial solution O(E)

Hamiltonian cycle

- G = (V, E) a connected, directed graph find a cycle that visits each vertex of G exactly once
- NP-complete

The Traveling Salesman Problem

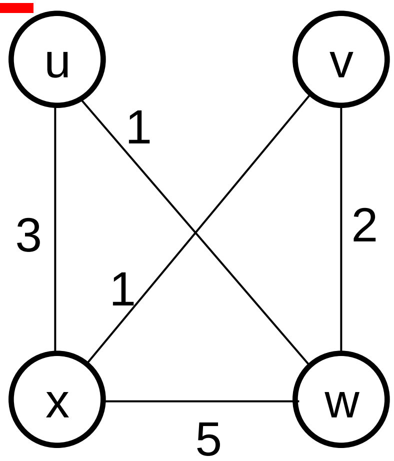
G = (V, E), |V| = n, vertices represent cities

Cost: $c(i, j)$ = cost of travel from city i to city j

Problem: salesman should make a tour (hamiltonian cycle):

- Visit each city only once
- Finish at the city he started from
- Total cost is minimum

TSP = tour with cost at most k



$\langle u, w, v, x, u \rangle$

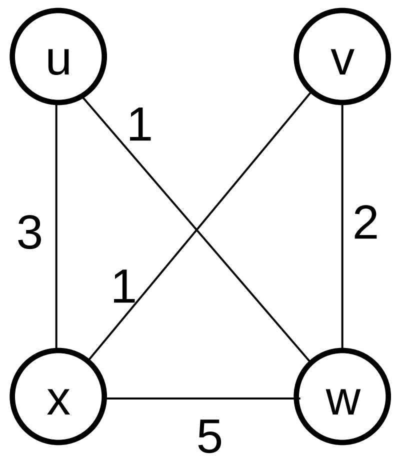
TSP ∈ NP

Certificate:

- Sequence of n vertices, cost
- E.g.: ⟨u, w, v, x, u⟩, 7

Verification:

- Each vertex occurs only once
- Sum of costs is at most k



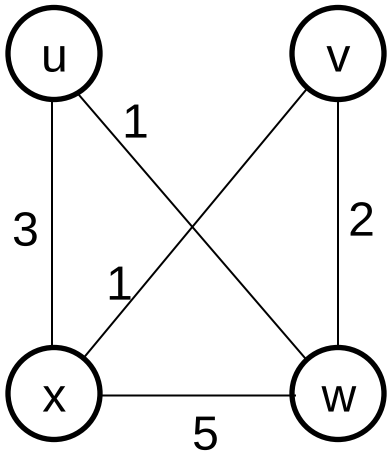
HAM-CYCLE $\leq_p$ TSP

Start with an instance of Hamiltonian cycle G = (V, E)

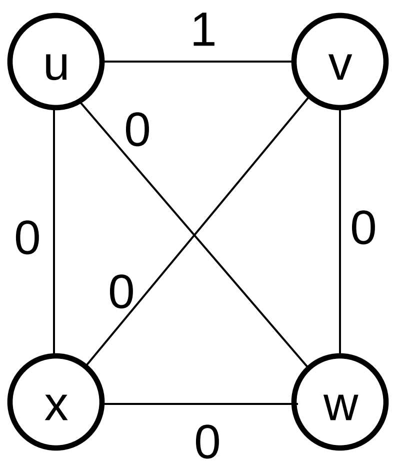
Form the complete graph G' = (V, E')

$$E' = \{(i, j): i, j \in V \text{ and } i \neq j\}$$

$$c(i, j) = \begin{cases} 0 & \text{if } (i, j) \in E \\ 1 & \text{if } (i, j) \notin E \end{cases}$$



TSP: $\langle G', c, 0\rangle$



G has a hamiltonian cycle $\Leftrightarrow$

G' has a tour of cost at most 0

HAM-CYCLE $\leq_p$ TSP

G has a hamiltonian cycle h

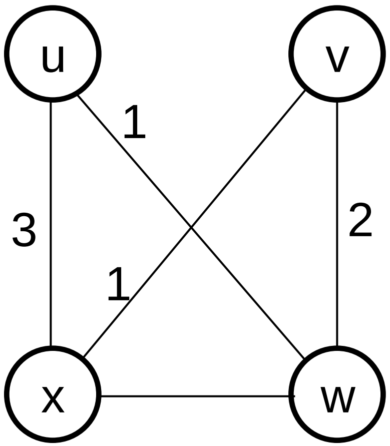
$\Rightarrow$ Each edge in $h \in E \Rightarrow$ has cost 0 in G'

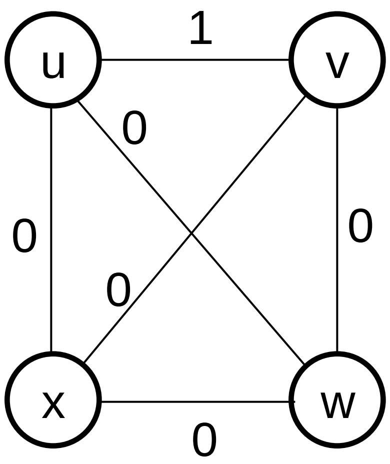
$\Rightarrow$ h is a tour in G' with cost 0

- G' has a tour h' of cost at most 0

$\Rightarrow$ Each edge on tour must have cost 0

$\Rightarrow$ h' contains only edges in E





- **Hamiltonian and Euler Graphs**

Hamiltonian Circuits

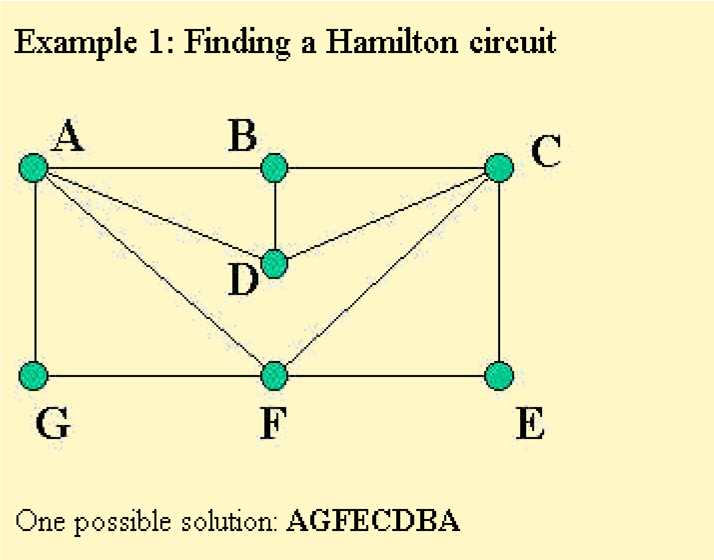
A ***Hamilton path*** in a graph *G* is a path which visits ever vertex in *G* exactly once. A ***Hamilton circuit*** (or ***Hamilton cycle***) is a cycle which visits every vertex exactly once, *except for the first vertex*, which is also visited at the end of the cycle.

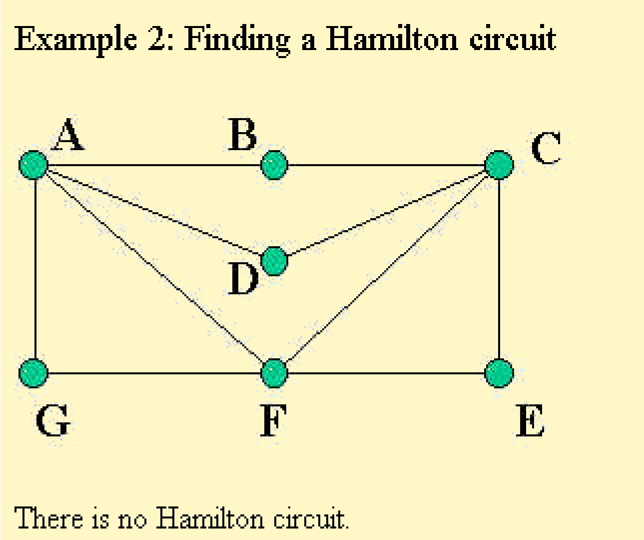
NOTE: The definition applies both to undirected as well as directed graphs of all types.

Hamiltonian Circuits

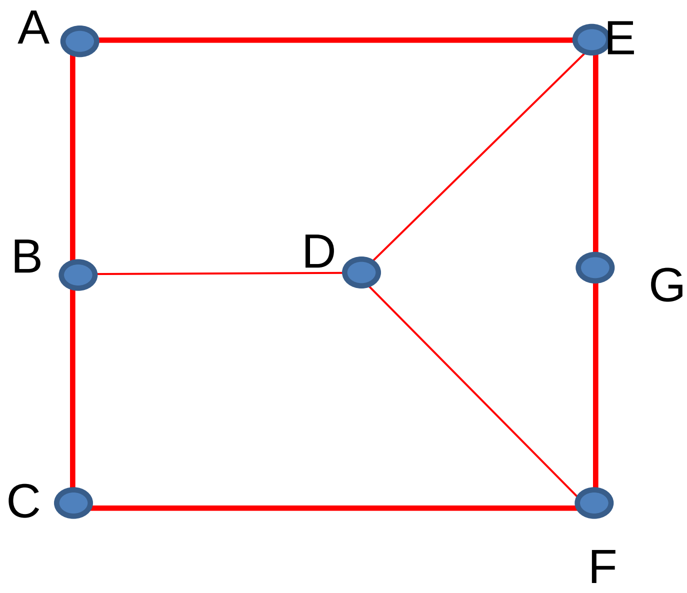
It is a so-called hard problem and there is no general condition for its existence

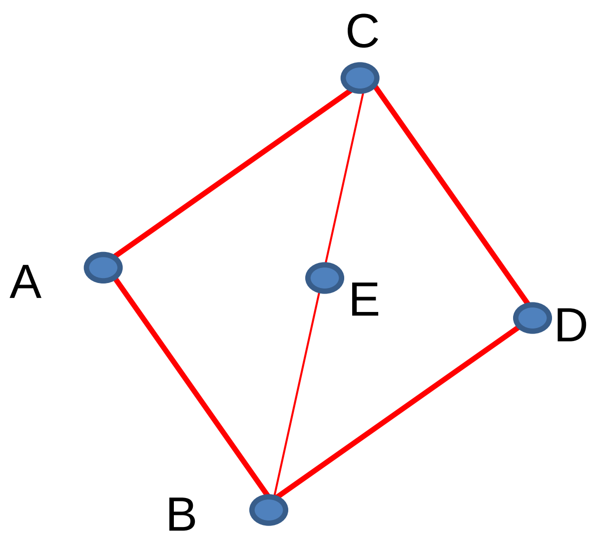
Hamiltonian Circuits





Hamiltonian Circuits





Euler Circuits

An ***Euler path*** in a graph *G* is a simple path containing every edge in *G*. An ***Euler circuit*** (or ***Euler cycle***) is a cycle which is an Euler path.

NOTE: The definition applies both to undirected as well as directed graphs of all types.

Euler Circuits

- A sequence of adjacent vertices and edges that
  1. starts and ends at the same vertex,
  2. uses every vertex of G <u>at least once</u>, and
  3. uses every edge of G <u>exactly once</u>

Euler Circuits

If a Graph has an Euler Circuit, every Vertex has Even Degree.

Contrapositive: if some vertex has odd degree, then the graph does not have an Euler circuit.

If every vertex of nonempty graph has even degree and if graph is connected, then the graph has an Euler circuit

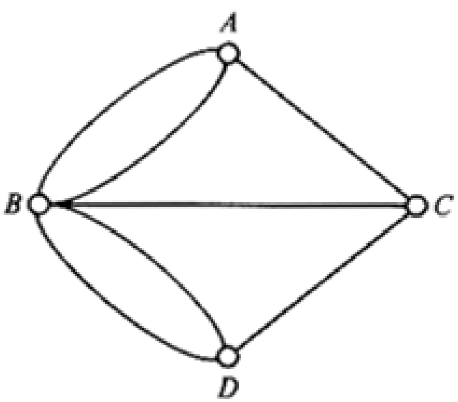
Euler Circuits

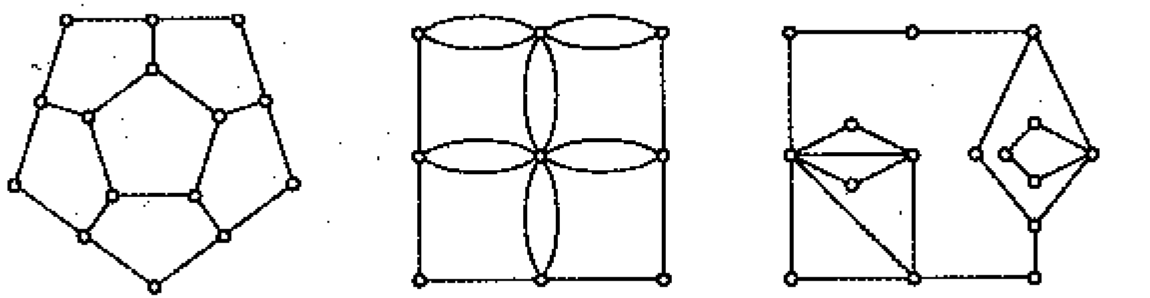
An undirected graph has at least one Euler circle i it is connected and has no vertices of odd degree. An Euler path exists exist iff there are no or zero vertices of odd degree.

A directed graph has at least one Euler circle i it is connected and for every vertex u in-degree(u)= out-degree(u). An Euler path exists exist iff there are exactly

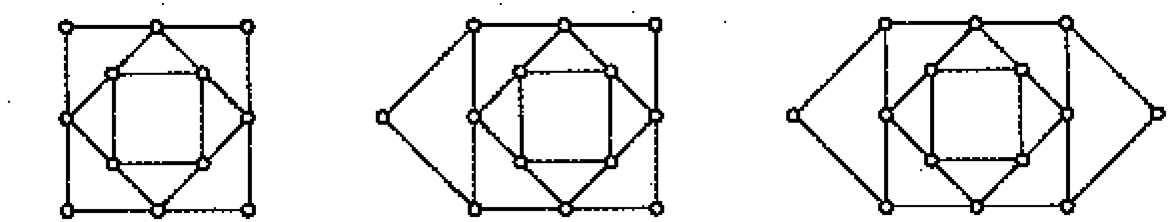
two vertices s,f for which the previous criterion does not hold and for which in-degree(s)=out-degree(s) -1 (starting vertex of the path) and in-degree(f)= out degree(f)+1 (final vertex of the path).

Euler Circuits





(a) (b) (c)



(a) (b) (c)

Special type of graphs-Euler Graphs

Eulerian (or Euler) circuits A circuit in a graph G where all the edges of the graph are traversed exactly once.

A graph which contains an Euler circuit is called an Euler Graph.

innovate achieve lead

Special type of graphs-Euler Graphs

Properties of Euler Graphs (Theorems)

(1) A given connected graph G is an Euler graph if and only if all vertices of G are of even degree.

(2) A connected graph G is an Euler graph if and only if it can be decomposed into circuits.

(ie) G should be a union of edge disjoint circuits.

Special type of graphs-Euler Graphs

Fleury's Algorithm

Step 1 choose a starting vertex u.

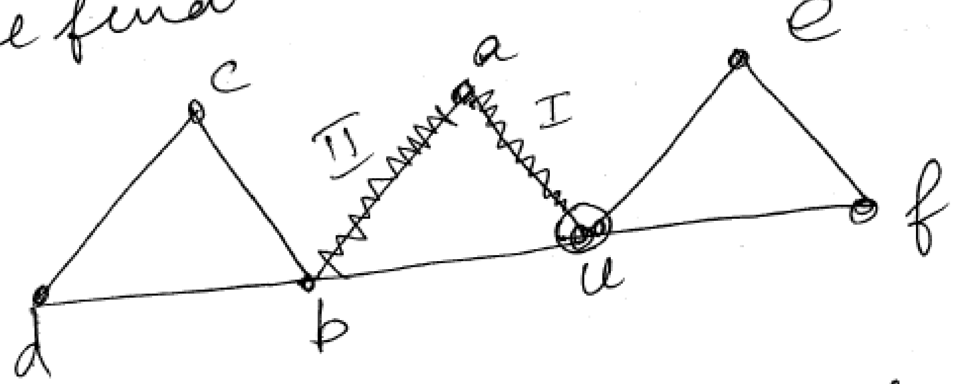
Step 2 At each stage, traverse any available edge, choosing a bridge only if there is no alternative

Step 3 After traversing each edge, erase it, erasing any vertices of degree 0 which result and then choose another available edge.

Step 4 Stop when there are no more edges.

Special type of graphs-Euler Graphs

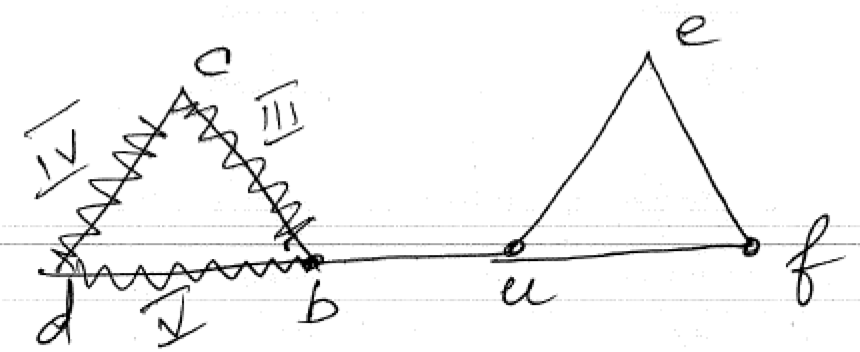
Example: Starting at vertex u, we find an Euler circuit in graph G



Step 1
We choose edge ua, followed by ab and the vertex a. Erasing these edges gives us the below graph.

Special type of graphs-Euler Graphs

Step 2 We cannot use the edge bu since it is a bridge, so we

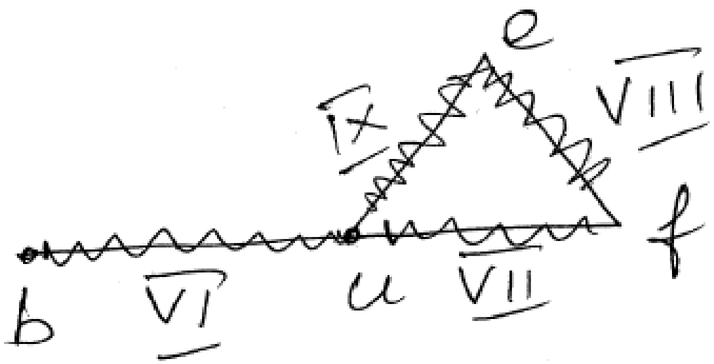


choose the edge bc, followed by cd and db.

Erasing these edges and the vertices c and d gives us the below graph

Special type of graphs-Euler Graphs

step 3.



We travel through bu, the bridge as there is no alternative. Then uf, fe and eu.

The Euler circuit is

uabcdbufeu.

innovate achieve lead

BITS Pilani
Pilani Campus

- **Backtracking**
- **Branch and Bound**

Backtracking

- Suppose a series of *decisions are to be made* , among various *choices,* where
  - Not enough information about what to choose
  - Each decision leads to a new set of choices
  - Some sequence of choices (possibly more than one) may be a solution to the problem
- **Backtracking** is a methodical way of trying out various sequences of decisions, until we find one that "works"

Backtracking

- Backtracking is used to solve problems in which a sequence of objects is chosen from a specified set so that the sequence satisfies some criterion.
- Backtracking is a modified depth-first search of a tree.
- Backtracking involves only a tree search.
- Backtracking is the procedure whereby, after determining that a node can lead to nothing but dead nodes, we go back ("backtrack") to the node's parent and proceed with the search on the next child.

Backtracking

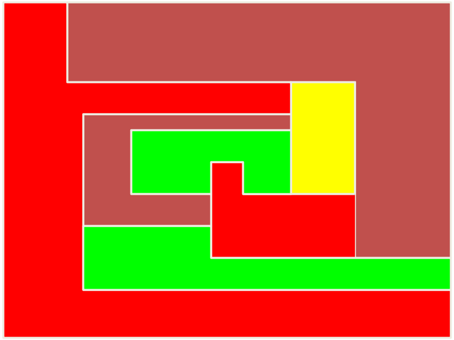
- We call a node nonpromising if when visiting the node we determine that it cannot possibly lead to a solution. Otherwise, we call it promising.
- In summary, backtracking consists of
  - Doing a depth-first search of a state space tree,
  - Checking whether each node is promising, and, if it is nonpromising, backtracking to the node's parent.
- This is called pruning the state space tree, and the subtree consisting of the visited nodes is called the pruned state space tree.

Solving a maze

- Given a maze, find a path from start to finish
- At each intersection, we have to decide between three or fewer choices:
  - Go straight
  - Go left
  - Go right
- we don't have enough information to choose correctly
- Each choice leads to another set of choices
- One or more sequences of choices may (or may not) lead to a solution
- Many types of maze problem can be solved with backtracking

Coloring a map

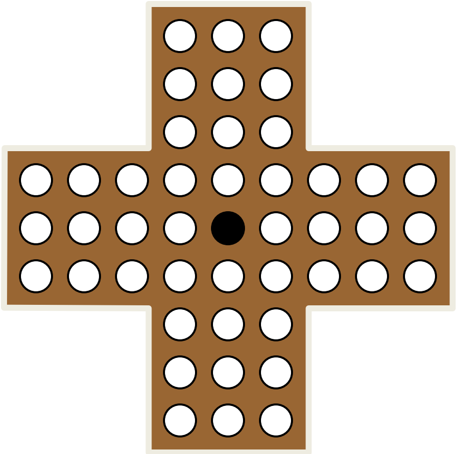
- We wish to color a map with not more than four colors
  - red, yellow, green, blue
- Adjacent countries must be in different colors



- don’t have enough information to choose colors
- Each choice leads to another set of choices
- One or more sequences of choices may (or may not) lead to a solution
- Many coloring problems can be solved with backtracking

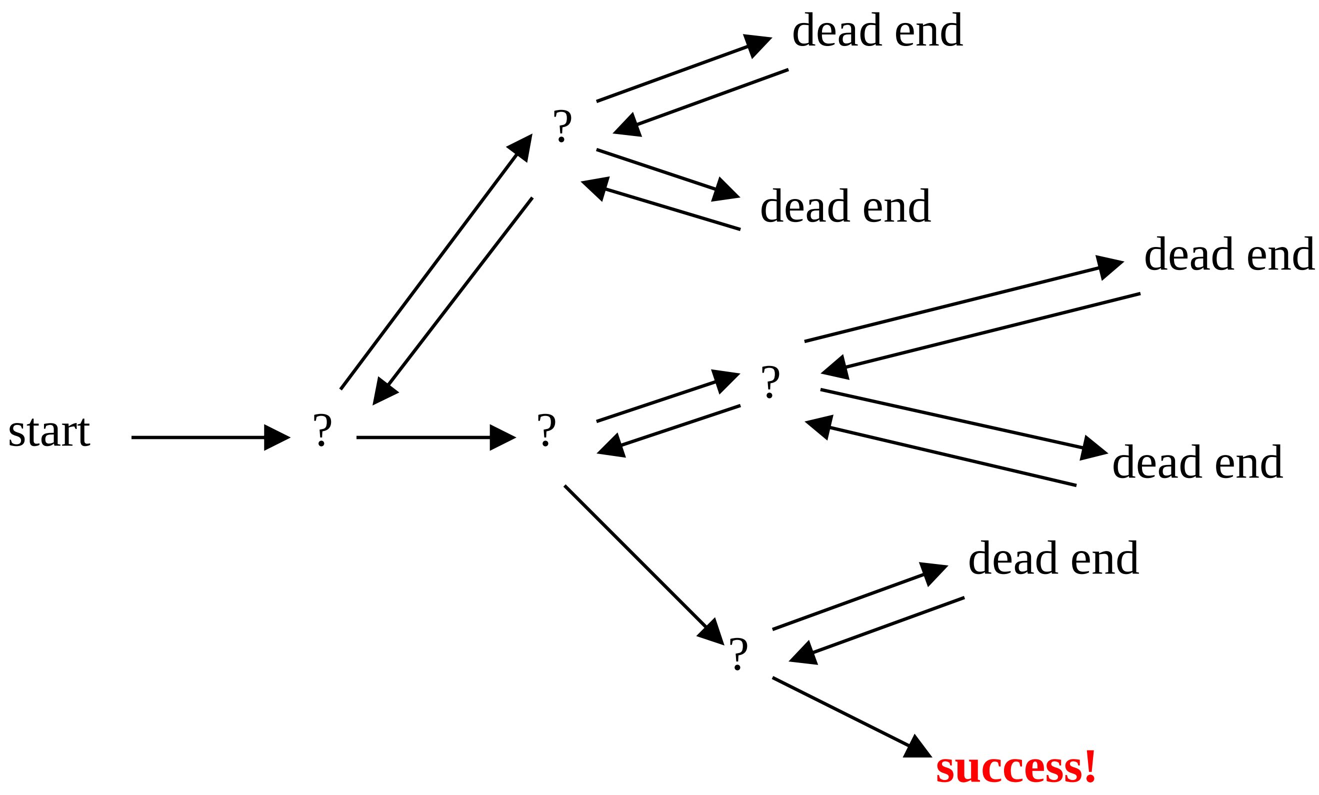
Solving a puzzle

- In this puzzle, all holes but one are filled with white pegs
- can jump over one peg with another
- Jumped pegs are removed
- The object is to remove all but the last peg



- don’t have enough information to jump correctly
- Each choice leads to another set of choices
- One or more sequences of choices may (or may not) lead to a solution
- Many kinds of puzzle can be solved with backtracking

Backtracking



Backtracking Algorithm

- Backtracking is really quite simple--we “explore” each node, as follows:

To “explore” node N:

1. If N is a goal node, return “success”
2. If N is a leaf node, return “failure”
3. For each child C of N,
   3.1. Explore C
      3.1.1. If C was successful, return “success”
4. Return “failure”

Sum of subsets

Problem: Given n positive integers $w_{1,}$ … w_n and a positive integer S. Find all subsets of $w_{1,}$ … w_n that sum to S.

Example:

n=3, m=6, and w_1=2, w_2=4, w_3=5

Solution:

{2,4}

Sum of subsets

```
Algorithm sumofsub(s,k,r)
    {
        //generate left child
        x[k] = 1;
        if (s + w[k] = m)
                then write (x[1:k]);//subset found
        else
                if (s + w[k] + w[k + 1] <= m)
                    then  sumof sub (s + w[k], k+1, r-w[k]);
        //gentrate right child
        if ((s + r – w[k] >= m) and (s + w[k+1] <= m))
                then
                        {           x[k]=0;
                                    sumofsub(s, k+1, r-w[k]);
                        }
    }
```

Sum of subsets

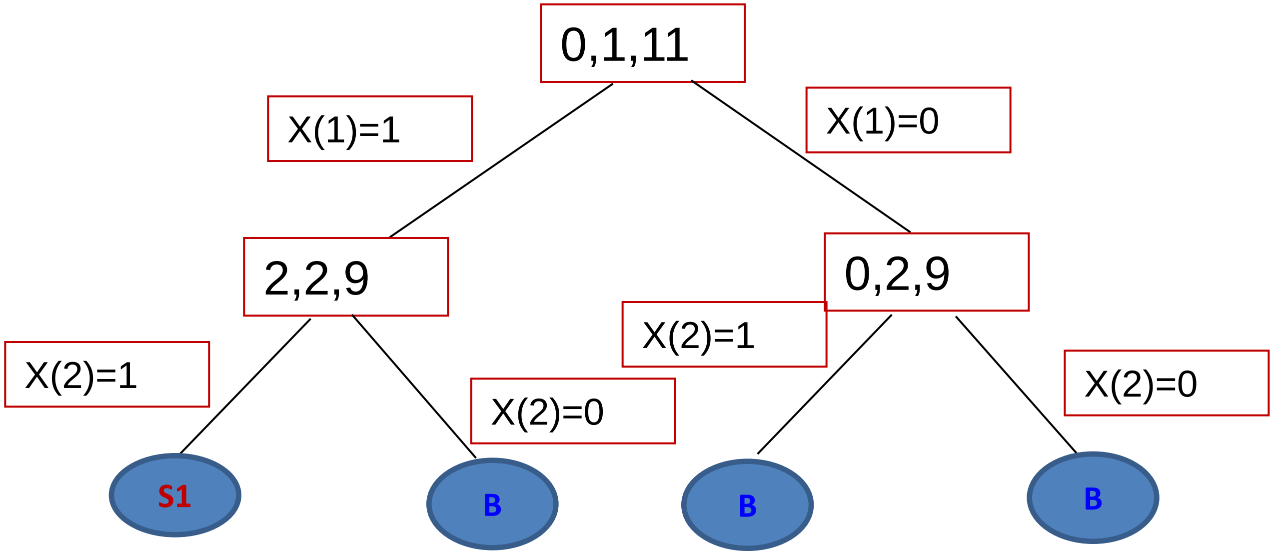
Example:

$n=3$, $m=6$, and $w_1=2$, $w_2=4$, $w_3=5$

$r = 2+4+5 = 11$

Initial call Sumofsub(0,1,11)

Solution set = 2+4=6
(1,1,0)



Sum of subsets

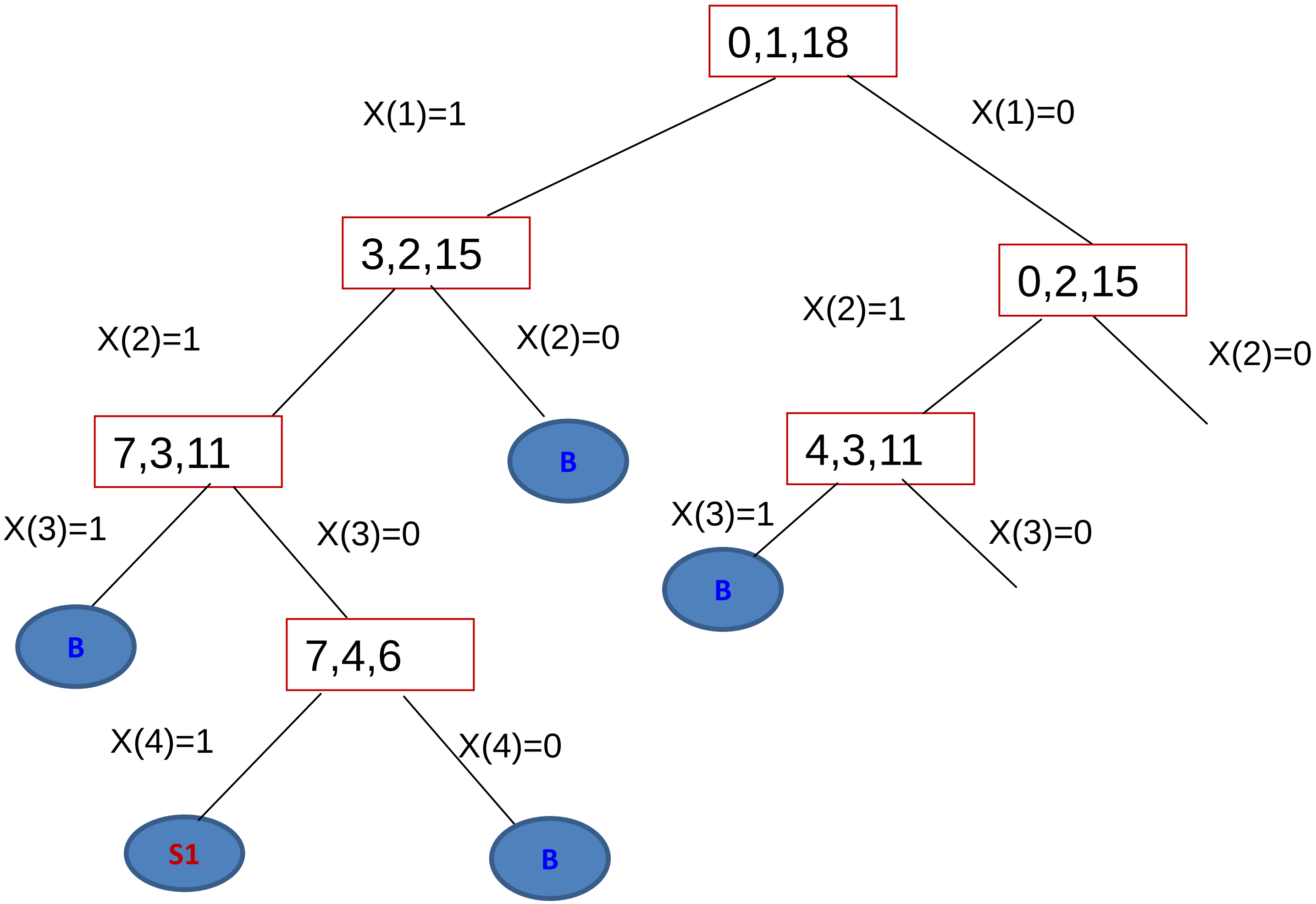
Suppose that $n = 4$, $m = 13$, and

$w_1 = 3$, $w_2 = 4$, $w_3 = 5$, $w_4 = 6$.

Find the solutions.

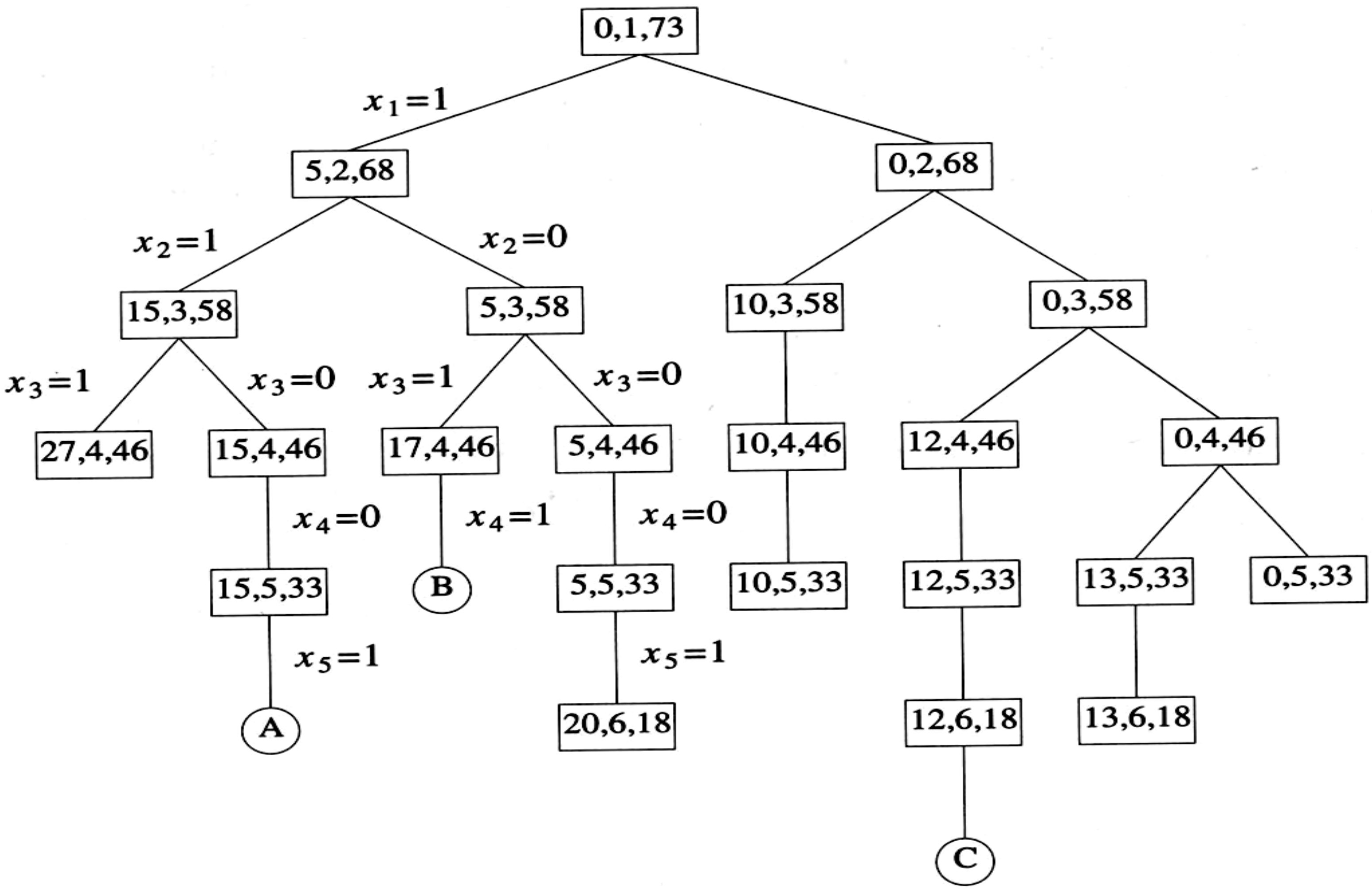
Initial call Sumofsub(0,1,18)

Sum of subsets



Sum of subsets

Solve for the solutions : n=6, w[1:6]={5,10,12,13,15,18}, m=30



Sum of subsets

innovate achieve lead

Suppose that $n = 5$, $W = 21$, and

$w_1 = 5$, $w_2 = 6$, $w_3 = 10$, $w_4 = 11$, and $w_5 = 16$.

Find the solutions.

Branch and Bound

- Where backtracking uses a depth-first search with pruning, the branch and bound algorithm uses a breadth-first search with pruning
- Branch and bound uses a queue as an auxiliary data structure

The Branch and Bound Algorithm

- Starting by considering the root node and applying a lower-bounding and upper-bounding procedure to it.
- If the bounds match, then an optimal solution has been found and the algorithm is finished.
- If they do not match, then algorithm runs on the child nodes.

The Branch and Bound Algorithm

1. **Branching:** Select an active subproblem Fi
2. **Pruning:** If the subproblem is infeasible, delete it.
3. **Bounding:** Otherwise, compute a lower bound b(Fi) for the subproblem.
4. **Pruning:** If b(Fi) >= U, the current best upperbound, delete the subproblem.
5. **Partitioning:** If b(Fi) <= U, either obtain an optimal solution to the subproblem (stop), or break the corresponding problem into further subproblems , which are added to the list of active subproblems

Example:
The Traveling Salesman Problem

Given:

A graph showing n number of cities connected by edges.

Cost of each edge is given using the cost matrix.

Aim of the problem:

Find the shortest path to cover all the cities and come back to the same city.

TSP Algorithm

1. Represent the above graph as weighted adjacency matrix.

   (Make the entry as ∞, if no path exists.)

2. Convert the above matrix into a cost reduction matrix.

   A cost reduction matrix is one in which at least one entry in each row and each column must be 0. For doing this, we need to reduce the minimum value from each element in each row and column

3. Calculate the reduced cost of the above matrix C(r).

4. For all the adjacent nodes of start node repeat the following

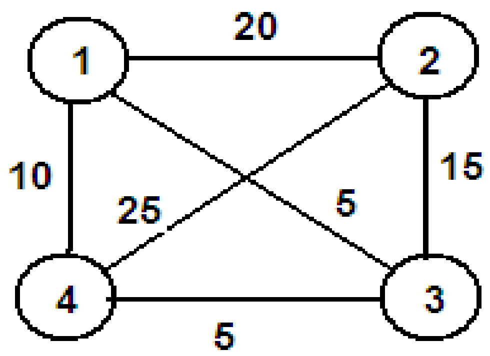
   a. Make all entries in the i^{th} row and j^{th} column to ∞

   b. Make A(j , 1) to ∞ (if 1 is the starting node)

   c. Find cost reduction matrix value C(s) = C(r) + A(i,j) + r

5. Select the path with minimum C(s) and repeat the step 4 with the corresponding cost reduction matrix as the input.

TSP (Example)



Find out the shortest path from node 1 and other nodes and return back to node 1

TSP (Example)

Step1. The Cost Matrix

	1	2	3	4
1	∞	20	5	10
2	20	∞	15	25
3	5	15	∞	5
4	10	25	5	∞

TSP (Example)

Step2 .The cost Reduction Matrix is

	1	2	3	4	
1	∞	5	0	5	5
2	5	∞	0	10	15
3	0	0	∞	0	5
4	5	10	0	∞	5
	0	10	0	0	40

C(r) = 40. Let this Matrix be A(i,j)

TSP (Example)

Step 4. The adjacent paths of 1 are (1,2) ,(1,3) , (1,4)

a. Path from (1,2)

- Make all entries in 1st row and 2nd column to ∞.
- Make A(2,1) to ∞
- Find cost reduction matrix value C(s) = C(r) + A(i,j) + r

	1	2	3	4	
1	∞	∞	∞	∞	0
2	∞	∞	0	10	0
3	0	∞	∞	0	0
4	5	∞	0	∞	0
	0	0	0	0	0

r = 0

Now , We calculate C(s)=40+5+0 = 45.

Let this matrix be A2(i,j)

TSP (Example)

Step 4. The adjacent paths of 1 are (1,2) ,(1,3) , (1,4)

b. Path from (1,3)

- Make all entries in 1st row and 3rd column to ∞.
- Make A(3,1) to ∞
- Find cost reduction matrix value C(s) = C(r) + A(i,j) + r

	1	2	3	4	
1	∞	∞	∞	∞	0
2	0	∞	∞	5	5
3	∞	0	∞	0	0
4	0	5	∞	∞	5
	0	0	0	0	**10**

r = 10

Now , We calculate C(s)=40+0+10 = 50

Let this matrix be A3(i,j)

TSP (Example)

Step 4. The adjacent paths of 1 are (1,2) ,(1,3) , (1,4)

b. Path from (1,4)

- Make all entries in 1^{st} row and 4^{th} column to ∞.
- Make A(4,1) to ∞
- Find cost reduction matrix value $C(s) = C(r) + A(i,j) + r$

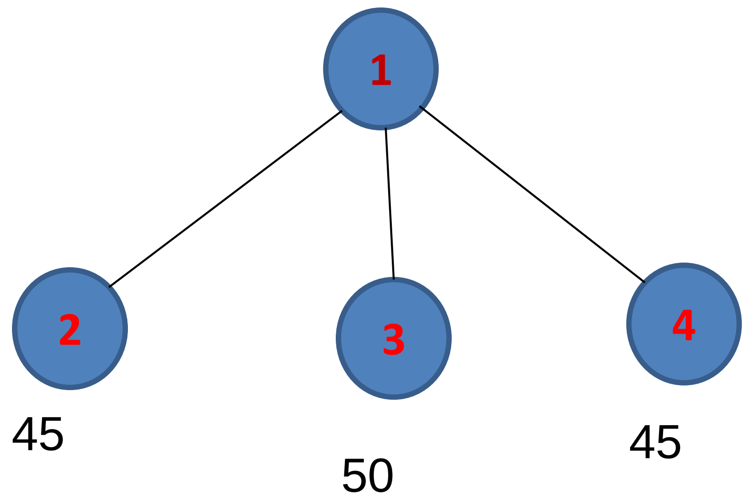
	1	2	3	4	
1	∞	∞	∞	∞	0
2	5	∞	0	∞	0
3	0	0	∞	∞	0
4	∞	10	0	∞	0
	0	0	0	0	0

r = 0

Now , We calculate C(s)=40+5+ 0 = 45

Let this matrix be A2(i,j)

TSP (Example)



Paths (1,2) and (1,4) are minimum. Any one path can be selected. If we select (1,2), 2 is the parent node and the cost for the paths (2,3) and (2,4) is to be calculated.

TSP (Example)

a. Path from (2,3)

- Make all entries in 2nd row and 3rd column to ∞.
- Make A(3,2) to ∞
- Find cost reduction matrix value C(s) = C(r) + A2(i,j) +

	1	2	3	4	
1	∞	∞	∞	∞	**0**
2	∞	∞	∞	∞	**0**
3	∞	∞	∞	0	**0**
4	0	∞	∞	∞	**5**
	0	**0**	**0**	**0**	**5**

r = 5

Now , We calculate C(s)=45+0+5 = 50

Let this matrix be A3(i,j)

TSP (Example)

innovate achieve lead

b. Path from (2,4)

- Make all entries in 2^{nd} row and 4^{th} column to ∞.
- Make A(4,2) to ∞
- Find cost reduction matrix value C(s) = C(r) + A4(i,j) +

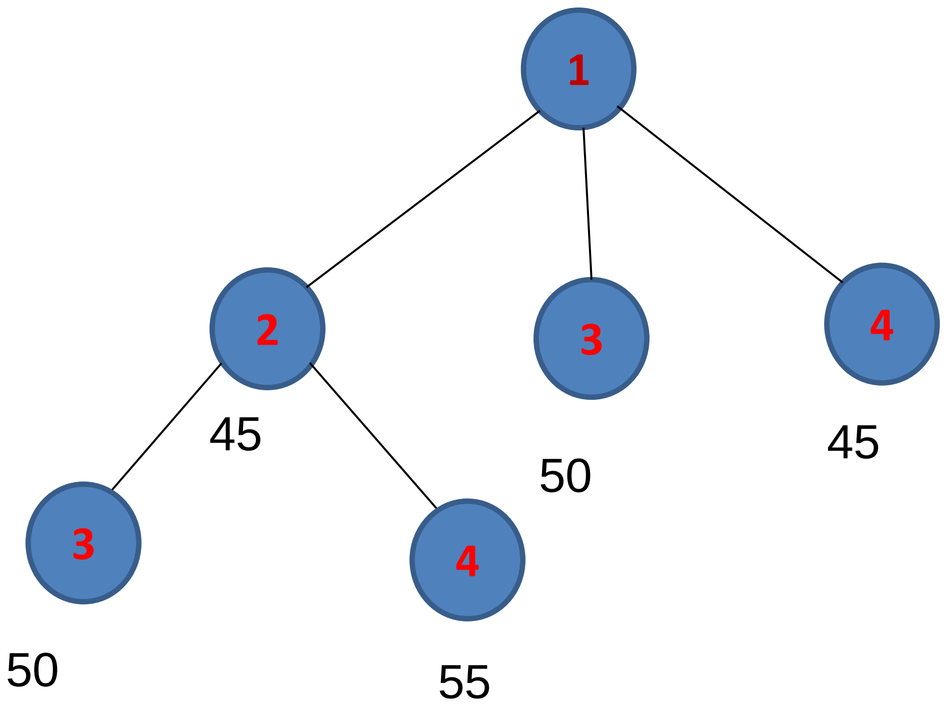
	1	2	3	4	
1	∞	∞	∞	∞	0
2	∞	∞	∞	∞	0
3	0	∞	∞	∞	0
4	∞	∞	0	∞	0
	0	0	0	0	0

r = 0

Now , We calculate C(s)=45+10+0 = 55

Let this matrix be A4'(i,j)

TSP (Example)



Path (2,3) is the minimum and hence it is selected. Cost of the path (3,4) is estimated next.

TSP (Example)

Path from (3,4)

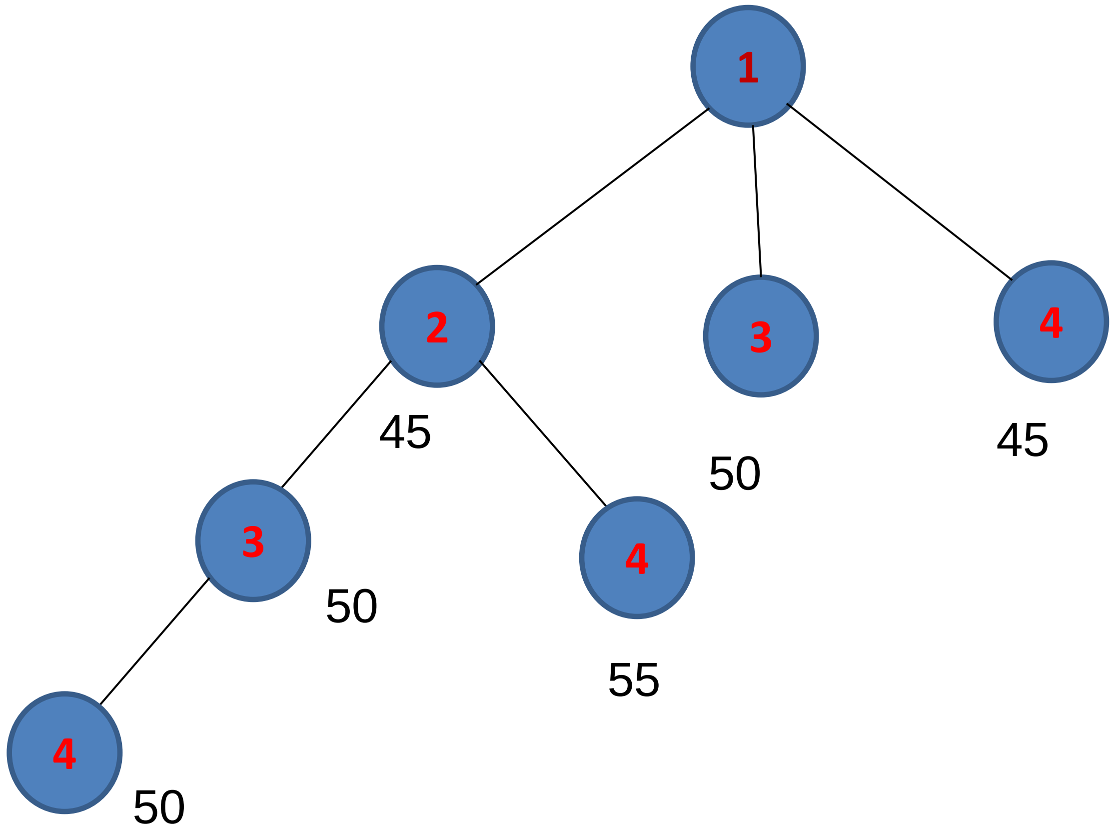
- Make all entries in 3rd row and 4th column to ∞.
- Make A(4,3) to ∞
- Find cost reduction matrix value C(s) = C(r) + A4'(i,j) +

	1	2	3	4	
1	∞	∞	∞	∞	0
2	∞	∞	∞	∞	0
3	∞	∞	∞	∞	0
4	∞	∞	∞	∞	0
	0	0	0	0	0

r = 0

Now , We calculate C(s)=50+0+0 = 50.

TSP (Example)



Path (2,3) is the minimum and hence it is selected. Cost of the path (3,4) is estimated next.

TSP (Example)

Therefore the solution to the above problem is

The shortest path → 1 → 2 → 3 → 4 → 1

The cost of the path is 50.

TSP (Example)

	1	2	3	4	5	6
1	∞	7	3	12	8	∞
2	3	∞	6	14	9	3
3	5	8	∞	6	18	5
4	9	3	5	∞	11	9
5	18	14	9	8	∞	18
6	∞	7	3	12	8	∞

Estimate and the reduced cost matrix and the lower bound value for the above cost matrix.