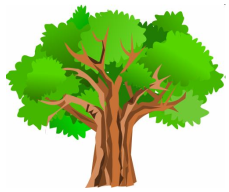
Recall: Trees



- The name of an important class of data objects
- Why is it called a tree?
  - Pieces of information are related by `branches'
- Special cases for special purposes
  - Binary trees, Binary search trees

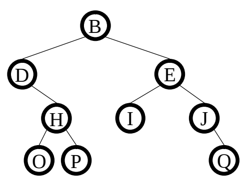
http://www.wpclipart.com

Trees

- What about a generalization?
  - i.e., something of which Trees is a special case
- Graphs
  - A graph is a collection of nodes along with a collection of branches
    - a branch connects 2 nodes
  - Terminology used:
    - Vertex (instead of node)
    - Edge (instead of branch)
  - A graph is a set of vertices along with a set of edges on those vertices

Graphs .

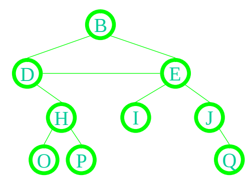
- Note: Every tree is a graph

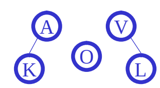


- Vertices: { B, D, E, H, I, J, O, P, Q }
- Edges: { (B,D), (B,E), (D,H), (H,O), (H,P), (E,I), (E,J), (J,Q) }

Graphs .

- Note: Not every graph is a tree

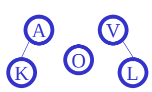




Vertex set: {A, K, L, O, V}
Edge set: {(A,K), (V,L)}

Directed graphs (Digraphs)

- Graphs where each edge is ordered (directed)
  - Arc: an edge with a direction



Vertex set: {A, K, L, O, V}

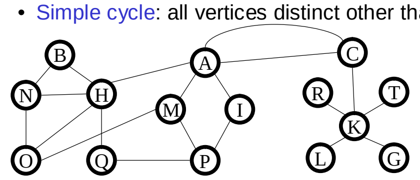
Edge set: {(A,K), (L,V)}

Note: Each edge is an ordered pair of vertices

- As opposed to Undirected graphs

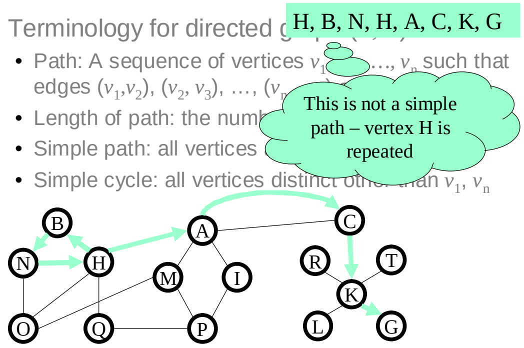
Directed graphs .

- Terminology for directed graph (V, E)
  - Path: A sequence of vertices $v_1, v_2, \ldots, v_n$ such that edges $(v_1,v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n) \in E$
  - Length of path: the number of arcs
  - Simple path: all vertices are distinct
  - Simple cycle: all vertices distinct other than v_1, v_n



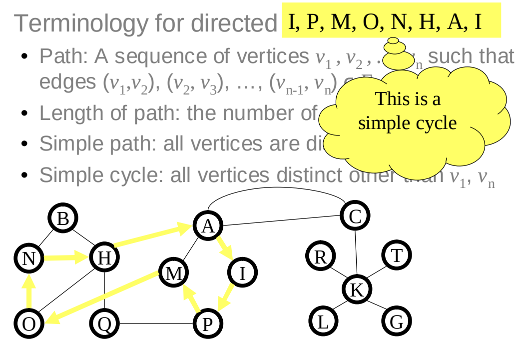
Directed graphs .

- Terminology for directed graph $G = (V, E)$
  - Path: A sequence of vertices $v_1, v_2, \ldots, v_n$ such that edges (v_1,v_2), (v_2, v_3), …, $(v_{n-1}, v_n) \in E$
  - Length of path: the number of edges
  - Simple path: all vertices are distinct
  - Simple cycle: all vertices distinct other than v_1, v_n



Directed graphs .

- Terminology for directed graph $G = (V, E)$
  - Path: A sequence of vertices $v_1, v_2, \ldots, v_n$ such that edges (v_1,v_2), (v_2, v_3), …, $(v_{n-1}, v_n) \in E$
  - Length of path: the number of edges
  - Simple path: all vertices are distinct
  - Simple cycle: all vertices distinct other than v_1, v_n



Representation?

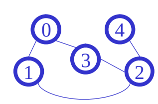
- For binary trees we discussed a pointer based representation
  - Each node had at most 2 branches
  - Not applicable to directed graphs in general
- For graphs, array based representations are therefore generally used instead
  1. Adjacency matrix representation
  2. Adjacency list representation

Adjacency Matrix Representation

- For the graph $G = (V, E)$ where $| V | = n$

  int AM [n, n]
  - One row for each vertex

  if $(v_i, v_k) \in E$, AM[i, k] = 1

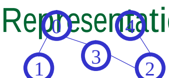


Vertex set: {0, 1, 2, 3, 4}

Edge set: {(0,1), (0,3), (1,2), (2,3), (2,4)}

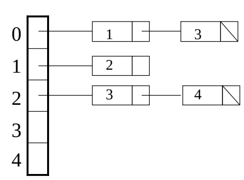
	0	1	2	3	4
0		1		1	
1			1		
2				1	1
3					
4					

Adjacency List Representation



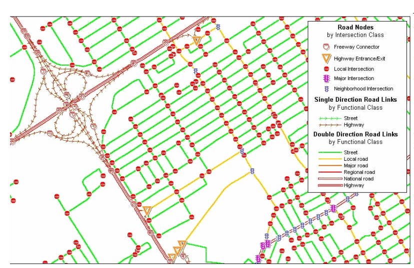
Vertex set: {0, 1, 2, 3, 4}

Edge set: {(0,1), (0,3), (1,2), (2,3), (2,4)}



A Graph Problem

- Given a road network



Thériault, Vandersmissen, Lee-Gosselin, Leroux, Journal of Geographic Information and Decision Analysis, 3(1):41-55, 1999

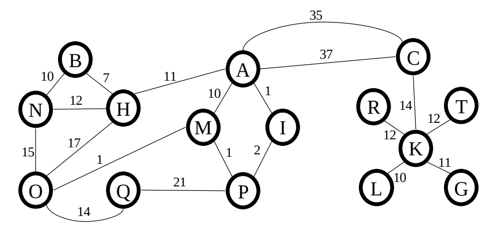
A Graph Problem .

- We can construct a graph of the road network
  - A vertex for each road intersection
  - One arc for each of the roads at the intersection, going from that intersection to the next intersection
- We can associate a value with each arc of the graph
  - “cost”, “weight”, “distance”
  - We will assume that these values are all non-negative
- And ask the following question:

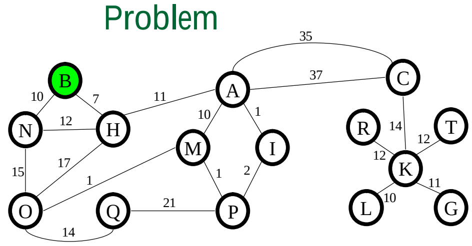
  “What is the shortest distance from one particular vertex to all the other vertices in the graph?”

Shortest Paths Problem

- “What is the shortest distance from a given vertex to all the other vertices in the graph?”
  - Single Source Shortest Paths Problem



Single Source Shortest Paths Problem



A	C	G	H	I	K	L	M	N	O	P	Q	R	T
33	68	93	22	34	82	92	37	10	38	36	52	94	94

Single Source Shortest Paths Problem

- We completed one vertex in each `iteration’
  - by moving to the vertex with the smallest additional distance from all completed vertices
  - This is an example of a greedy algorithm
  - Greedy: Do what seems best right now and hope that this yields a globally optimal result
- We have `discovered’ Dijkstra’s Algorithm
  - Edsger Dijkstra (1930-2002)
    - 1972 Turing Award winner

Dijkstra’s Single Source Shortest Paths

Let $V = \{1, 2, 3, \ldots, n\}$, with source being vertex 1
E specified with costs in adjacency matrix M[n, n]

Initialize set *Completed* = {1}, D[i] = M[1, i], $n \geq i > 1$

for (i = 2; i <= n; i++) {
Determine vertex v ∊ *V* – *Completed* with minimum D[v]
Insert v into *Completed*
for each vertex x ∊ *V* – *Completed*
D[x] = min (D[x], D[v]+M[v, x])
}

Number of comparisons? O ($|V|^2$)

Food for thought

Given a graph $G = (V, E)$ where $|V| = n$

Adjacency matrix M(n, n)

entries are distances

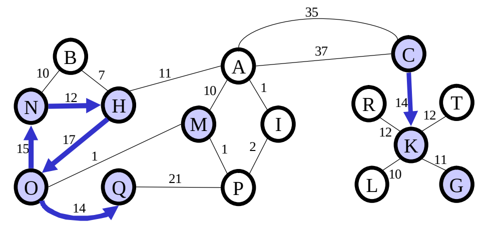
non-existent edges represented by 0s

What is M^2 ?

Undirected Graphs

- $G = (V, E)$ where each edge is an unordered pair of vertices
  - Adjacency matrix is a symmetric matrix
- Additional terminology:
  - Connected graph: A graph is said to be connected if every pair of vertices is connected
    - For every pair of vertices, there is (at least one) path connecting them
  - Subgraph: $G' = (V', E')$ is a subgraph of $G = (V, E)$
    - $V' \subseteq V$
    - E' contains some edges $(u, v) \in E$ for which both $u, v \in V'$

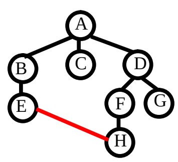
Subgraph



Trees (again)

A simple path that starts and ends with the same vertex

- A tree is an undirected graph
  - any two vertices are connected by exactly one simple path (connected)
  - without cycles (acyclic)



- If you add an edge to a tree, a cycle is created

A Graph Problem

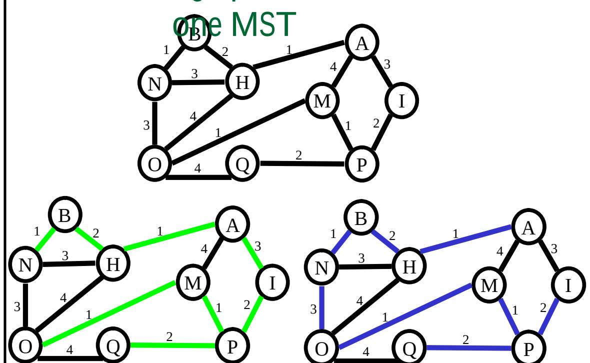
Given a connected, undirected graph G in which each edge has an associated cost

find a subgraph of G that

- includes all the vertices of G
- is a tree (i.e., a connected, acyclic graph)
- has the lowest total cost

`Minimum Spanning Tree Problem'

A graph can have more than one MST

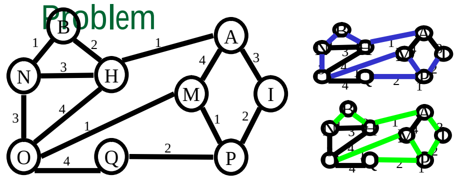


Claim

`If (u, v) is an edge of lowest cost, then there exists an MST that includes it as an edge.'

We can construct the MST by greedily add edges using this property

Minimum Spanning Tree Problem



V	A B H I M N O P Q
U	A B H I M N O P Q
MST	(Q, P) (P, M) (M, O) (P, I) (O, N) (N, B) (B, H) (H, A)

Minimum Spanning Tree Algorithm (Prim)

Let $V = \{1, 2, 3, \ldots, n\}$

E specified with costs in adjacency matrix M[n, n]

Initialize sets $U = \{1\}$, $MST = \{\}$

while ($U \neq V$) {

Determine lowest cost (u, v) s. t. $u \in U$ and $v \in V - U$

$MST = MST \cup \{(u,v)\}$

$U = U \cup \{(u,v)\}$

}

A Graph Problem

Given a connected, undirected graph G in which each edge has an associated cost

and a specified starting vertex, v

find a Hamiltonian path of minimal total cost that starts and ends at vertex v

i.e., a simple cycle that includes each vertex of G exactly once

`Travelling Salesman Problem'

Travelling Salesman Problem

- Until now, we have used greedy algorithms for our graph problems
- Known greedy algorithms for the Travelling Salesman Problem are not guaranteed to give optimal solutions
  - or even to work for all graphs

A Greedy Approach to TSP?

Sort the edges (on increasing cost)

Consider the edges one by one

What would happen if it is added to the path?

Does it make any vertex degree > 2?

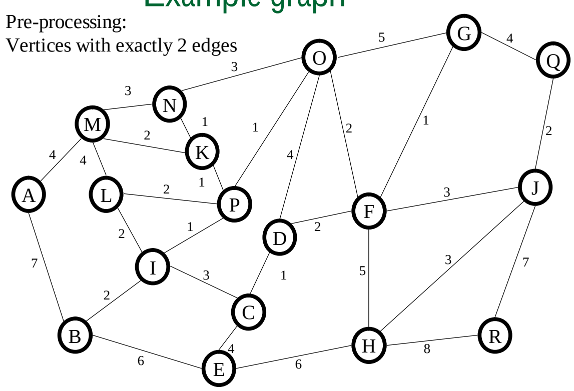
Does it complete a cycle?

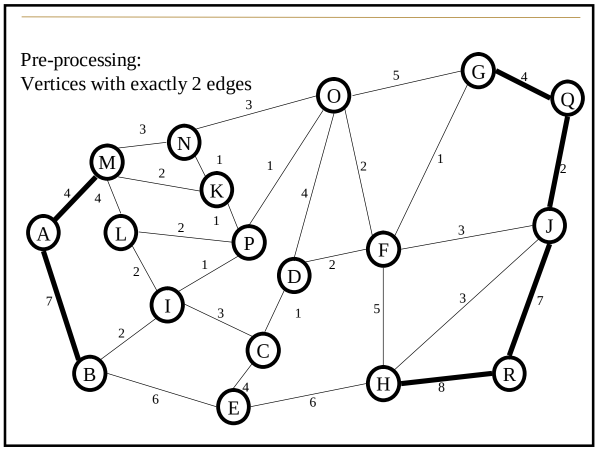
If not, add it and continue

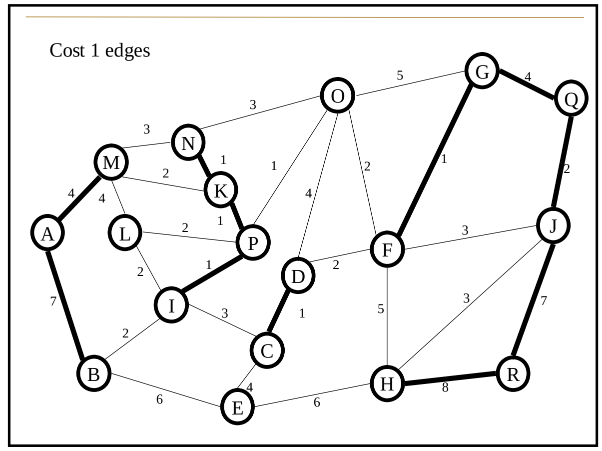
Example graph

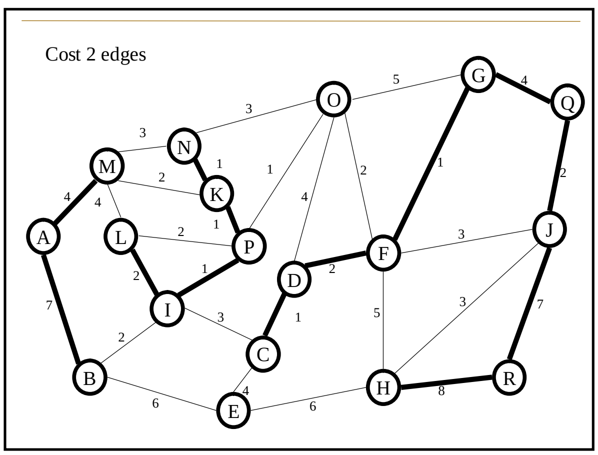
Pre-processing:

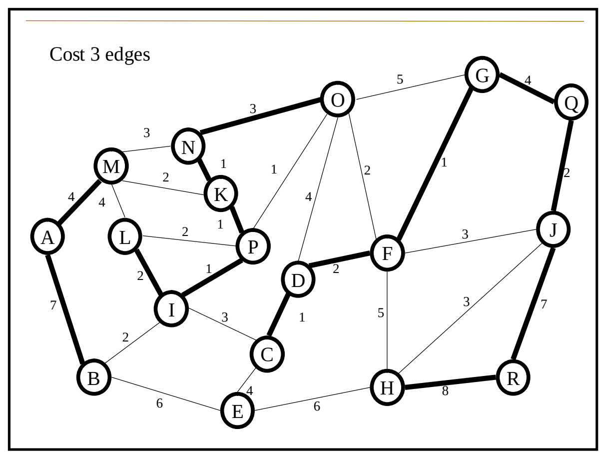
Vertices with exactly 2 edges

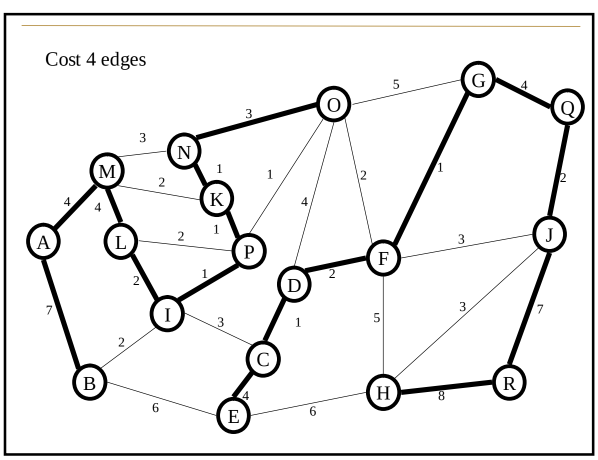


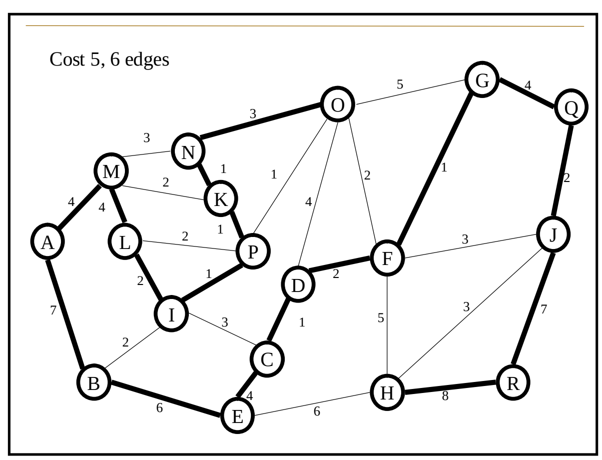

Pre-processing:
Vertices with exactly 2 edges


Cost 1 edges

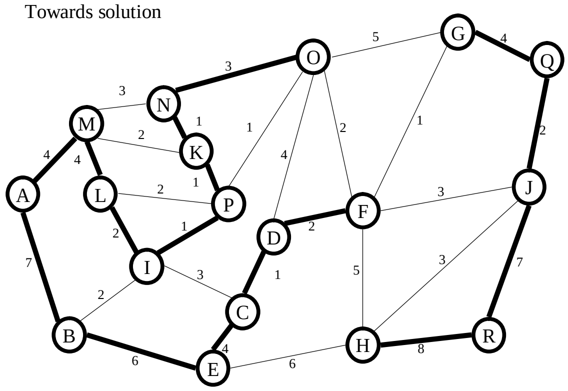

Cost 2 edges


Cost 3 edges


Cost 4 edges

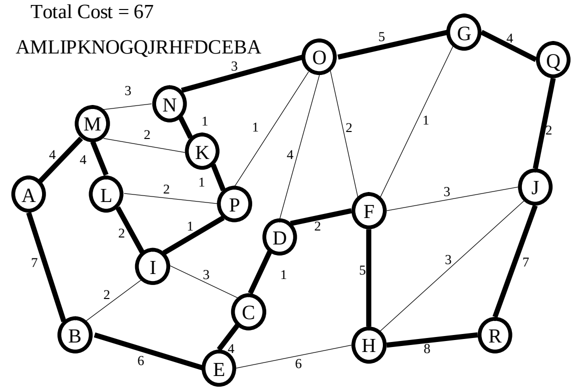

Cost 5, 6 edges

Towards solution



Total Cost = 67

AMLIPKNOGQJRHFDCEBA

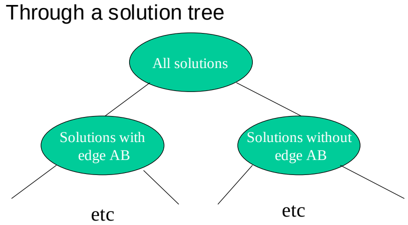


Travelling Salesman Problem

- Until now, we have used greedy algorithms for our graph problems
- Known greedy algorithms for the Travelling Salesman Problem are not guaranteed to give optimal solutions
  - or even to work for all graphs
- We could enumerate and compare all possible solutions
  - Expensive O(n!)
- So, we will use another technique: Branch and bound
  - A general technique to find optimal solutions for optimization problems

Branch and Bound

- Systematically enumerate all solutions
  - Through a solution tree



- Need: a lower bound for the cost of solutions in a branch of the solution tree

TSP: Lower bound on solution cost

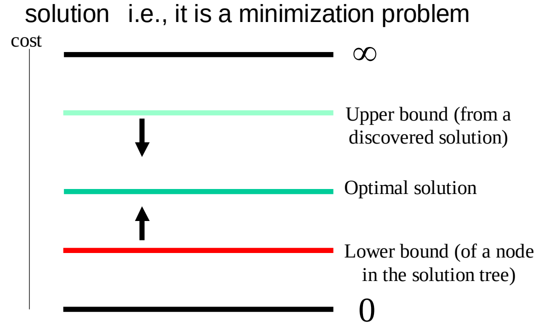
$$\text{Cost of any solution S} = \sum_{e \in S} \text{Cost}(e)$$

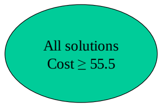
$$= \frac{1}{2} \sum_{v \in V} \text{Costs of the 2 edges through vertex } v \text{ in S}$$

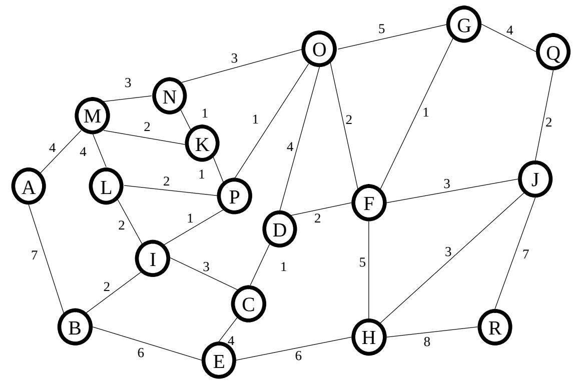
$$\geq \frac{1}{2} \sum_{v \in V} \text{Costs of 2 lowest cost edges through vertex } v$$

Upper and Lower Bounds

In TSP, we want to find the minimum cost solution i.e., it is a minimization problem

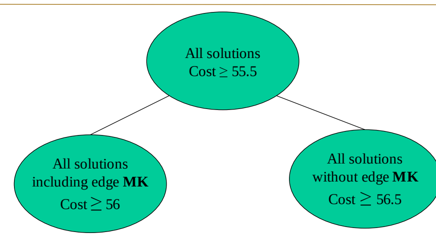


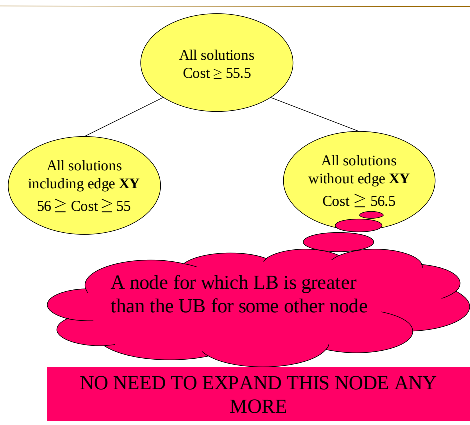


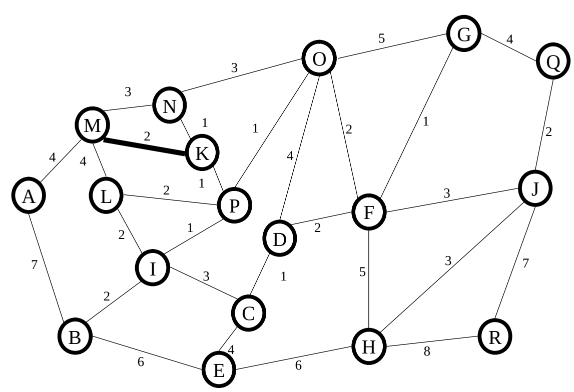


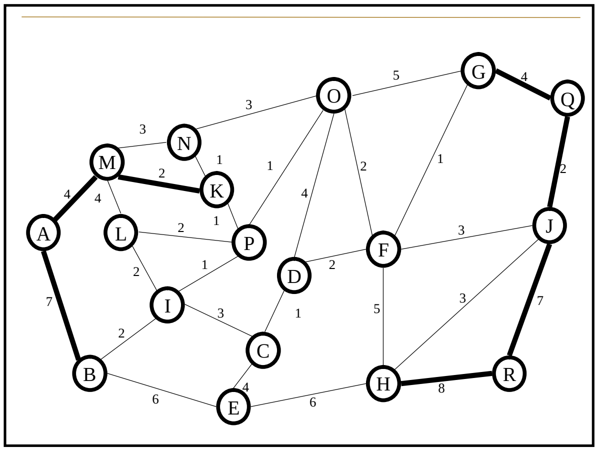
Sum across of all vertices of 2 lowest cost edges at that vertex $= \dfrac{111}{2} = 55.5$

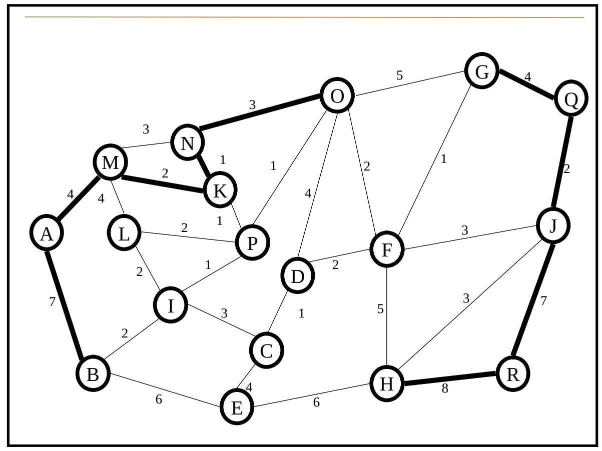
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
4	2	1	1	4	1	1	3	1	2	1	2	2	1	1	1	2	7
7	6	3	2	6	2	4	5	2	3	1	2	3	3	2	1	4	8
	7	4	4	6	2	5	6	2	3	2	4	4	3	3	1		
					3		8	3	7			4		4	2		
					5									5			

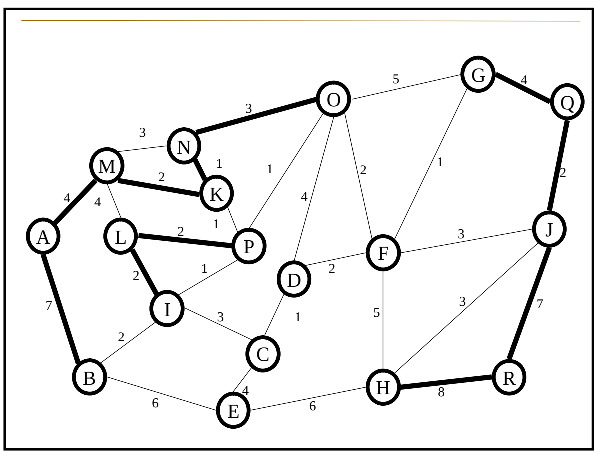


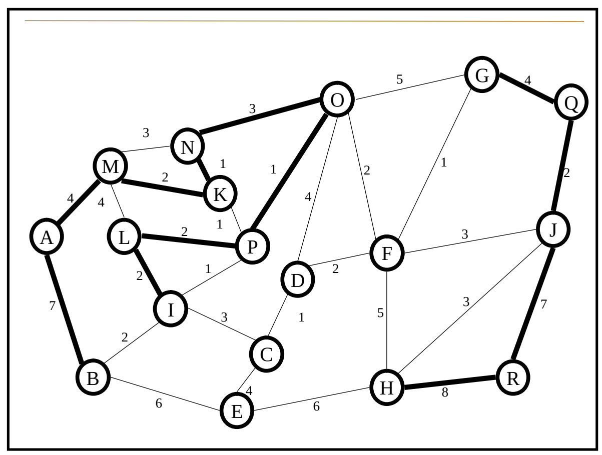


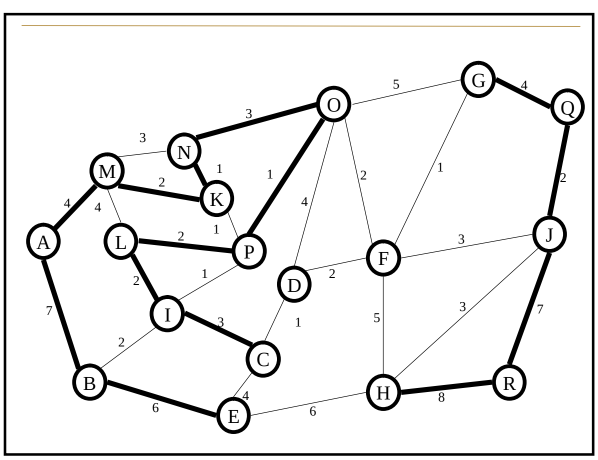


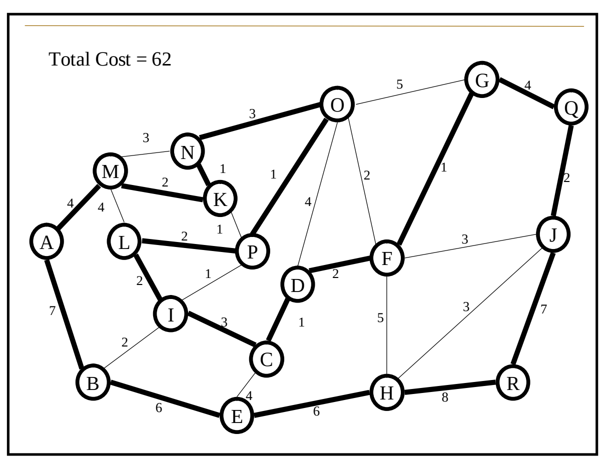


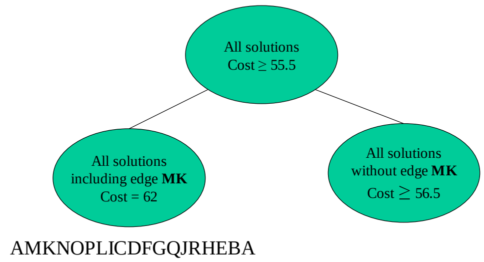


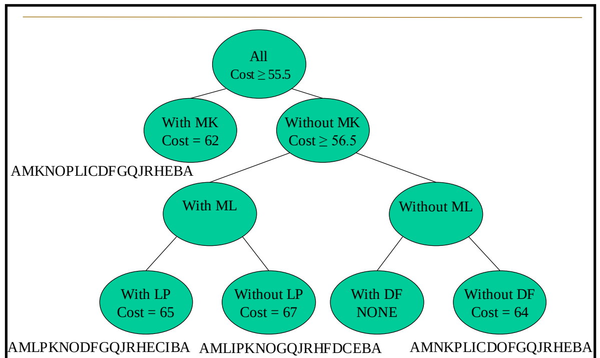







Total Cost = 62



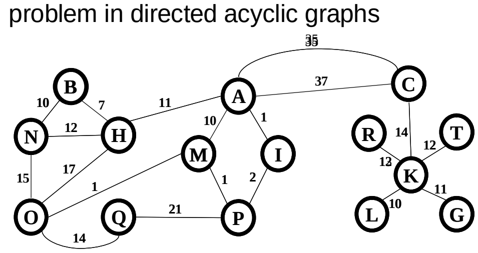


Travelling Salesman Problem

- Known greedy algorithms for the Travelling Salesman Problem are not guaranteed to give optimal solutions
  - Or to work for all graphs
- We could enumerate and compare all possible solutions: Brute Force Approach
  - Expensive
- So, we could use another technique: Branch and bound
- Or yet another: Dynamic Programming

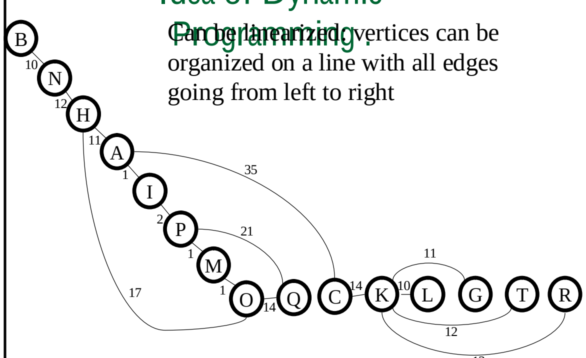
Idea of Dynamic Programming

- Consider single source shortest paths problem in directed acyclic graphs

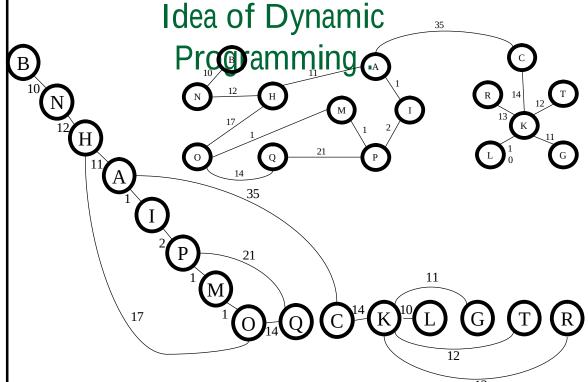


Idea of Dynamic Programming

Can be linearized: vertices can be organized on a line with all edges going from left to right



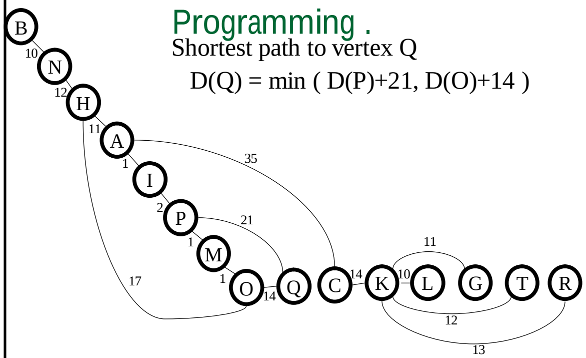
Idea of Dynamic Programming



Idea of Dynamic Programming .

Shortest path to vertex Q

D(Q) = min (D(P)+21, D(O)+14)



Idea of Dynamic Programming ..

Initialize distances: D(source) = 0, all others D[i] = ∞

for (each vertex v, in linearized order)

$$D[v] = \min_{(u,v) \text{ in } E} (D[u] + cost(u,v))$$

Solving a collection of subproblems

- starting with the simplest subproblem
- using the answers of simpler subproblems to solve later subproblems

Dynamic Programming for TSP

- What are the subproblems?
- Suppose that we started with vertex 1 and have reached vertex *j* in our solution
  - What information is needed to extend this tour?
    - The vertices that have been visited so far
  - For a subset of vertices S ⊆ {1, 2, …, *n*} including 1, and *j* ∈ S, let C(S, *j*) be the length of the shortest path visiting each vertex in S exactly once, starting at 1 and ending at *j*

Dynamic Programming for TSP.

- Smaller subproblem?
  - What should be picked as the previous (to *j*) vertex?
    - Some vertex *i* ∈ S such that

$C(S, j) = \min_{i \text{ in } S,\, i \neq j} C(S - \{ j \}, i) + cost(i, j)$

- Smallest subproblem?

  C({1}, 1) = 0
- Order the subproblems

  on cardinality of S

C(S, *j*): the length of the shortest path visiting each vertex in S exactly once, starting at 1 and ending at *j*

Dynamic Programming for TSP..

C ({1}, 1) = 0

for s = 2 to *n* {

for all subsets S ⊆ {1, 2, …, *n*} of size s containing 1 {

C(S, 1) = ∞

for all j ∈ S, j ≠ 1

C(S, *j*) = min{ C(S - {*j*}, i) + cost(i, j): i ∈ S, i ≠ j}}

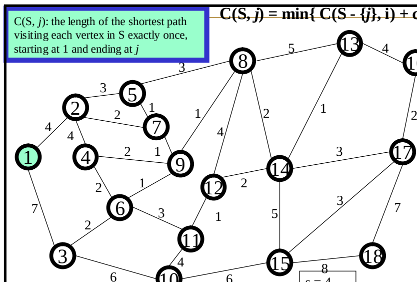
}

return $\min_j$ C({1,…., *n*}, *j*) + cost(*j*, 1)

C(S, *j*): the length of the shortest path visiting each vertex in S exactly once, starting at 1 and ending at *j*

C(S, *j*): the length of the shortest path visiting each vertex in S exactly once, starting at 1 and ending at *j*

C(S, *j*) = min{ C(S - {*j*}, i) + cost(i, j): i ∈ S, i ≠ j}



s = 2

C({1, 2}, 2) = 4

C({1, 3}, 3) = 7

s = 3

C({1, 2, 4}, 4) = 8

C({1, 2, 5}, 5) = 7

C({1, 2, 7}, 7) = 6

C({1, 3, 6}, 6) = 9

C({1, 3, 10}, 10) = 13

s = 4

C({1, 2, 4, 6}, 6) = 10

C({1, 2, 4, 9}, 9) = 10

C({1, 2, 5, 7}, 5) = 7

C({1, 2, 5, 7}, 7) = 8

C({1, 2, 5, 8}, 8) = 10

C({1, 2, 7, 9}, 9) = 7

C({1, 3, 4, 6}, 4) = 11

C({1, 3, 6, 9}, 9) = 10

C({1, 3, 6, 11}, 11) = 12

C({1, 3, 10, 11}, 11) = 17

C({1, 3, 10, 15}, 15) = 19

s = 5

C({1, 2, 3, 4, 6}, 2) = 15

C({1, 2, 3, 4, 6}, 3) = 12

C({1, 2, 4, 6, 9}, 6) = 11

C({1, 2, 4, 6, 9}, 9) = 11

C({1, 2, 4, 6, 11}, 11) = 13

C({1, 2, 4, 7, 9}, 4) = 9

C({1, 2, 4, 7, 9}, 7) = 11

C({1, 2, 4, 8, 9}, 8) = 11

C({1, 2, 5, 7, 9}, 9) = 9

C({1, 2, 5, 8, 9}, 9) = 11

C({1, 2, 5, 8, 12}, 12) = 14

C({1, 2, 5, 8, 13}, 13) = 15

C({1, 2, 5, 8, 14}, 14) = 12

C({1, 2, 6, 7, 9}, 6) = 8

C({1, 2, 7, 8, 9}, 8) = 8

C({1, 2, 3, 4, 6}, 2) = 15

C({1, 2, 3, 4, 6}, 3) = 12

C({1, 3, 4, 6, 9}, 4) = 12

C({1, 3, 4, 6, 9}, 9) = 13

C({1, 3, 6,7, 9}, 7) = 11

C({1, 3, 6,8, 9}, 8) = 11

C({1, 3, 6, 10,11}, 6) = 20

C({1, 3, 6, 10,11}, 10) = 16

C({1, 3, 6, 11, 12}, 12) = 13

C({1, 3, 10,11, 12}, 12) = 18

Dynamic Programming for TSP..

C ({1}, 1) = 0
for s = 2 to n {
for all subsets S ⊆ {1, 2, …, n) of size s containing 1 {
C(S, 1) = ∞
for all j ∈ S, j ≠ 1
C(S, j) = min{ C(S - {j}, i) + cost(i, j): i ∈ S, i ≠ j}}
}
return min$_j$ C({1,…., n}, j) + cost(j, 1)

Time complexity: $O(n^2 2^n)$

Space complexity: $O(n 2^n)$