Brute Force

A straightforward approach, usually based directly on the problem's statement and definitions of the concepts involved

Simple Examples:

1. Computing a^n (a > 0, n a nonnegative integer)
2. Computing n!
3. Multiplying two matrices
4. Searching for a given value in a list
5. Naïve sorting algorithms
   e.g. Selection Sort

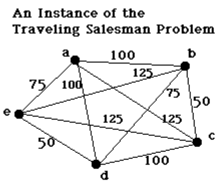
Exhaustive Search

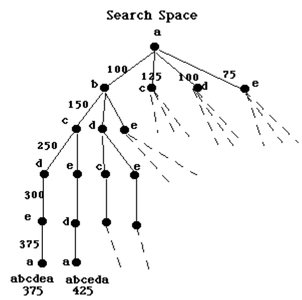
- A brute force solution to a problem involving search for an element with a special property.

- Method:
  - Generate a list of all potential solutions to the problem in a systematic manner.
  - Evaluate potential solutions one by one disqualifying infeasible ones and, for an optimization problem, keeping track of the best one found so far.
  - When search ends, announce the solution(s) found.

Example 1: Traveling Salesman Problem

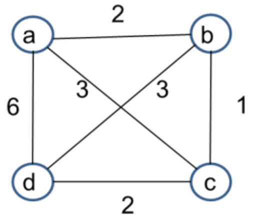
- Given *n* cities with known distances between each pair, find the shortest tour that passes through all the cities exactly once before returning to the starting city





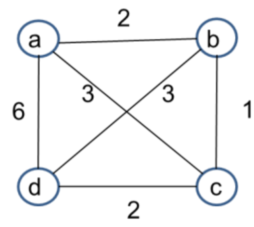
Example 1: Traveling Salesman Problem

- Given *n* cities with known distances between each pair, find the shortest tour that passes through all the cities exactly once before returning to the starting city
- Alternatively: Find shortest *Hamiltonian circuit* in a weighted connected graph
- Example:



Example 1: Traveling Salesman Problem

- Nearest Neighbor (greedy) does not guarantee an optimal solution!



TSP by Exhaustive Search

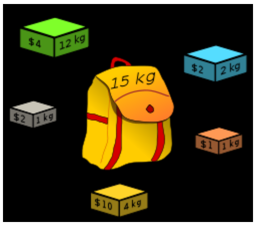
Tour	Cost
a→b→c→d→a	2+3+7+5 = 17
a→b→d→c→a	2+4+7+8 = 21
a→c→b→d→a	8+3+4+5 = 20
a→c→d→b→a	8+7+4+2 = 21
a→d→b→c→a	5+4+3+8 = 20
a→d→c→b→a	5+7+3+2 = 17

How many **distinct** tours? 5 cities? N cities?

Example 2: 0-1 Knapsack Problem

- Given n items:
  - weights: w1 w2 … wn
  - values: v1 v2 … vn
  - a knapsack of capacity W
- Find most valuable subset of the items that fit into the knapsack
- Example: Knapsack capacity W=16

item	weight	value
1	2	$20
2	5	$30
3	10	$50
4	5	$10



Knapsack Problem by Exhaustive Search

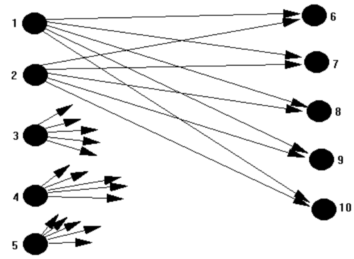
Subset	Total weight	Total value
{1}	2	$20
{2}	5	$30
{3}	10	$50
{4}	5	$10
{1,2}	7	$50
{1,3}	12	$70
{1,4}	7	$30
{2,3}	15	$80
{2,4}	10	$40
{3,4}	15	$60
{1,2,3}	17	not feasible
{1,2,4}	12	$60
{1,3,4}	17	not feasible
{2,3,4}	20	not feasible
{1,2,3,4}	22	not feasible

Example 3: The Assignment Problem

There are n people who need to be assigned to n jobs, one person per job. The cost of assigning person i to job j is C[i,j]. Find an assignment that minimizes the total cost.

	Job 0	Job 1	Job 2	Job 3
Person 0	9	2	7	8
Person 1	6	4	3	7
Person 2	5	8	1	8
Person 3	7	6	9	4

How many assignments are there?



Assignment Problem by Exhaustive Search

$$\mathbf{C} = \begin{matrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{matrix}$$

Assignment (col.#s)	Total Cost
1, 2, 3, 4	9+4+1+4=18
1, 2, 4, 3	9+4+8+9=30
1, 3, 2, 4	9+3+8+4=24
1, 3, 4, 2	9+3+8+6=26
1, 4, 2, 3	9+7+8+9=33
1, 4, 3, 2	9+7+1+6=23
	etc.

Final Comments on Exhaustive Search

- Exhaustive-search algorithms run in a realistic amount of time only on very small instances
- In some cases, there are much better alternatives!
  - Euler circuits
  - shortest paths
  - minimum spanning tree
  - assignment problem
- In many cases, exhaustive search or its variation is the only known way to get exact solution

Combinatorial Objects: Generating Permutations and Subsets

- Conceptually the easiest way to think about these objects is recursively
- This leads to relatively straightforward implementations.

Generating Permutations

- Think top down: Suppose you can generate permutations of any set of (n-1)-elements.
- The given a set of n-elements use the black box on the n-1 elements.
- Given {a,b,c} how would this work to generate the permutations in lexicographic order? Write the pseudo code.

Generating Subsets - Bitstrings

- Think top down: Suppose you can generate all bitstrings of length (n-1).
- Use the black box on the bitstrings of length (n-1) to get the bitstrings of length n.
- How? Write the pseudo code.

Final Comments on Exhaustive Search

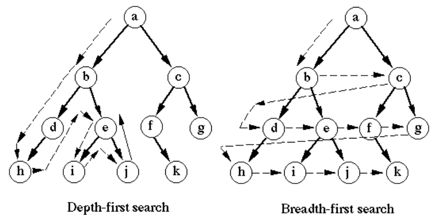
- Exhaustive-search algorithms run in a realistic amount of time only on very small instances
- In some cases, there are much better alternatives!
  - Euler circuits
  - shortest paths
  - minimum spanning tree
  - assignment problem
- In many cases, exhaustive search or its variation is the only known way to get exact solution

Graphs

- G=(V,E) V= Vertex Set; E= Edge Set
- Directed: edges ordered pairs undirected: unordered pairs
- Path: sequence of vertices $v_0, v_1, \ldots, v_k$
  where for i=1,…k, v_i-1 is adjacent to v_i (or sequence of edges)
- Simple path: no vertex or edge repeated except possibly the first
- Cycle: path where $v_0 = v_k$ k>1
- G is connected if there is a path between every pair of vertices
- Connected Component of G is a subgraph of G that is connected
- Adjacency Matrix → entry (i,j) = 1 if there is an edge=(i,j) else 0
- Adjacency List → each vertex has a list of the adjacent vertices

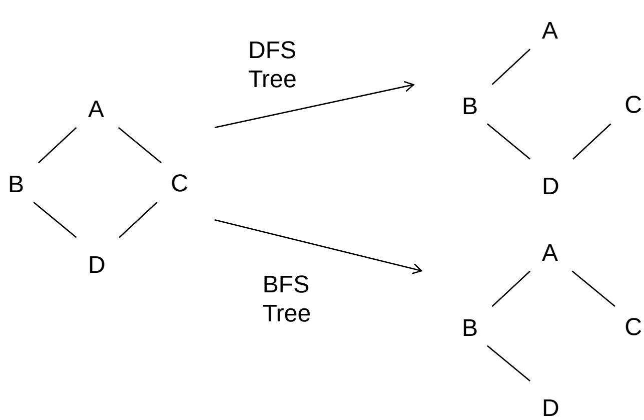
Graph Traversal Algorithms

- Many problems require processing all graph vertices (and edges) in systematic fashion.
- Computer representation of graphs contains little direct information about the overall structure.
- Graph traversal algorithms:
  - Depth-first search (DFS)
    - explore deep first
    - Stack -- LIFO
  - Breadth-first search (BFS)
    - explore all "neighbors" first
    - Queue – FIFO



- If G is a tree then preorder, inorder, and postorder are all depth first search algorithms!

Example: DFS vs BFS "spanning trees"



Depth-First Search (DFS)

- Visits graph's vertices by always moving away from last visited vertex to unvisited one, backtracks if no adjacent unvisited vertex is available.
- Uses a stack (explicitly or implicitly)
  - a vertex is pushed onto the stack when it's reached for the first time
  - a vertex is popped off the stack when it becomes a dead end, i.e., when there is no adjacent unvisited vertex
- "Redraws" graph in tree-like fashion (with tree edges and back edges for undirected graph)
  - *Tree edge – an edge that is traversed by the algorithm*
  - *Back edge – an edge in the original graph that is not part of the tree that is traversed in visiting all the nodes of the graph. Edges that lead to vertices that have already been visited!*

Depth First Search Algorithm

- Recursive marking algorithm
- Initially every vertex is unmarked

```
DFS(i: vertex)
  mark i as visited;
  for each j adjacent to i do
    if j is unmarked then DFS(j)
end{DFS}
```

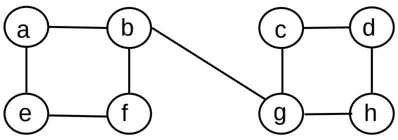
Marks all vertices reachable from i

DFS

```
DFS(G)
       Initialize cnt = 0 and visited(v) = false
       for all v ∈ V
              if not visited(v) then dfs(v)
dfs(v)
       visited(v) = true
       previsit(v); cnt = cnt + 1; pre(v)= cnt;
       for each edge (v,u) ∈ E
              if not visited(u) then dfs(u)
       postvisit(v); cnt = cnt + 1; post(v) = cnt
```

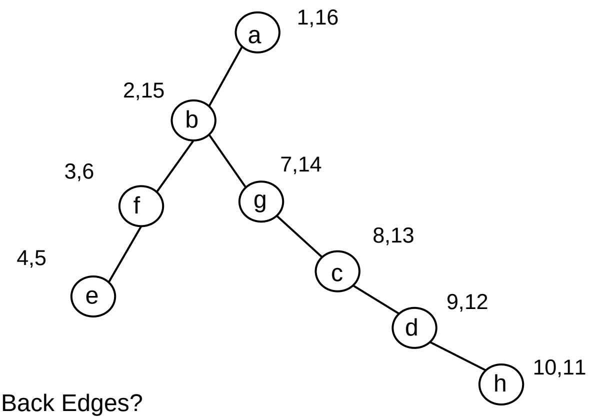
- The order of edges is usually determined by the data structure.
- Our convention when doing by hand, order is by the order of the symbols, e.g. alphabetical!

Example: DFS traversal of undirected graph



DFS (a)

Example: DFS traversal of undirected graph

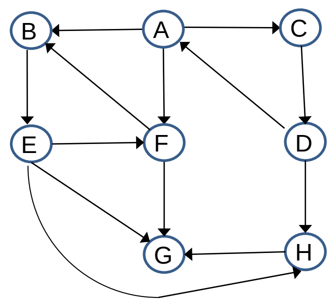


Back Edges?

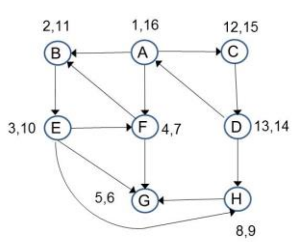
Directed Graphs – things to know

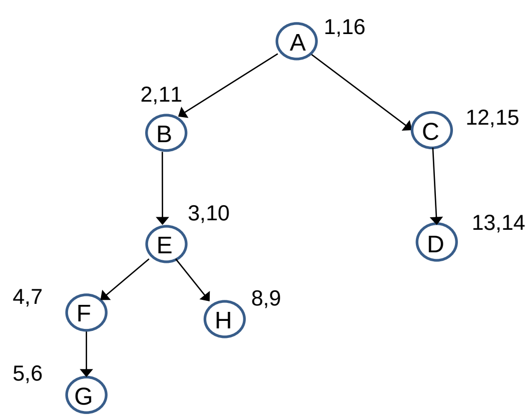
- Directed Graph or Digraph: every edge has a direction.
  - Adjacency matrix – may not be symmetric. Adjacency list – only put edge in one not two of the edge lists
- When constructing the depth-first search *forest* you can get 4 different types of edges
  - tree edges
  - non tree edges of the following types
    - » forward edges – to descendants other than children
    - » back edges – to ancestors
    - » cross edges – to anywhere else

Example: DFS traversal of directed graph



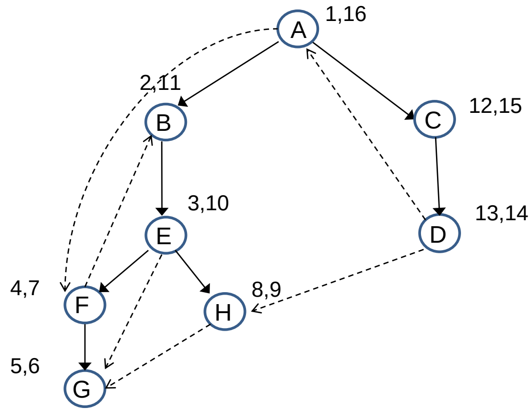
Example: DFS traversal of directed graph





Draw in the Back (2), Forward(2), and Cross Edges(2) – labeling appropriately.

Example: DFS traversal of directed graph



Draw in the Back (2), Forward(2), and Cross Edges(2) – labeling appropriately.

Classification of edges by a DFS of a directed graph

- Tree edges – part of the DFS forest - edges that involve visiting a vertex for the first time
- Forward edges – edges from a node to a non-child descendent in a DFS tree – the descendent has already been visited
- Backward edges – edges to an ancestor in the DFS tree
- Cross edges – edges to a vertex that is neither an ancestor or a descendent

Pre and Post numbers

Let (u,v) be an edge from u to v. Let u_{pre}, u_{post} and v_{pre}, v_{post} denote the pre and post numbers for vertex u and v respectively

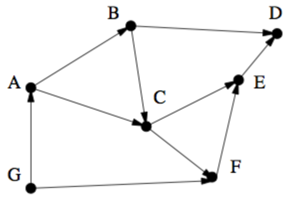
- [u_{pre}, u_{post}] [v_{pre}, v_{post}] : the interval for v is contained entirely in the interval for u $\rightarrow$ the edge from u to v is either a *tree* or a *forward* edge, said another way v is a *descendent* of u. u_{pre}, v_{pre}, v_{post}, u_{post}
- [u_{pre}, u_{post}] [v_{pre}, v_{post}] : the intervals are entirely disjoint $\rightarrow$ the edge from u to v is a cross edge, neither is a descendent of the other. v_{pre}, v_{post}, u_{pre}, u_{post},
- [u_{pre}, u_{post}] [v_{pre}, v_{post}] : the interval for u is contained entirely in the interval for v $\rightarrow$ the edge from u to v is a *backward* edge, v is a *ancestor* of u. v_{pre}, u_{pre}, u_{post}, v_{post},

Notes on DFS

- DFS can be implemented with graphs represented as:
  - adjacency matrices: $\Theta(V^2)$
  - adjacency lists: $\Theta(|V|+|E|)$
- Yields two distinct ordering of vertices:
  - order in which vertices are first encountered (pushed onto stack)
  - order in which vertices become dead-ends (popped off stack)
- Applications:
  - checking connectivity, finding connected components (strong and weak)
  - checking acyclicity
  - finding articulation points and biconnected components
  - searching state-space of problems for solution (AI)

Directed Acyclic Graphs: Topological Sorting

- A *DAG*: a directed acyclic graph, i.e. a directed graph with no (directed) cycles   (Constraint modeling)
- Topological Sorting: Vertices of a DAG can be linearly ordered so that for every edge its starting vertex is listed before its ending vertex



Applications of Topological Sorting

- Instruction scheduling
- Ordering of formula cell evaluation when recomputing formula values in spreadsheets
- Determining the order of compilation tasks to perform in makefiles
- Data serialization and resolving symbol dependencies in linkers
- Determine order to load tables with foreign keys in databases

- Project Evaluation and Review Technique, commonly abbreviated PERT, is a statistical tool, used in project management, which was designed to analyze and represent the tasks involved in completing a given project.
- Critical Path Method

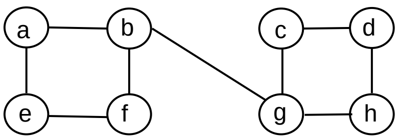
DFS-based Algorithm for Topological Sort

- Perform DFS traversal, noting the order vertices are popped off the traversal stack
- Reverse order (list vertices with highest post number first) solves topological sorting problem
- Back edges encountered?→ NOT a dag!

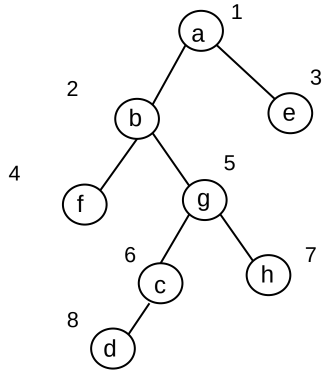
Breadth-first search (BFS)

- Visits graph vertices by moving across to all the neighbors of last visited vertex
- Instead of a stack, BFS uses a queue
- Similar to level-by-level tree traversal
- “Redraws” graph in tree-like fashion (with tree edges and cross edges for undirected graph)

Example of BFS traversal of undirected graph



Example: BFS traversal of undirected graph



Cross Edges ?

Pseudocode of BFS

```
ALGORITHM  BFS(G)
    //Implements a breadth-first search traversal of a given graph
    //Input: Graph G = ⟨V, E⟩
    //Output: Graph G with its vertices marked with consecutive integers
    //in the order they have been visited by the BFS traversal
    mark each vertex in V with 0 as a mark of being "unvisited"
    count ← 0
    for each vertex v in V do
        if v is marked with 0
            bfs(v)

    bfs(v)
    //visits all the unvisited vertices connected to vertex v by a path
    //and assigns them the numbers in the order they are visited
    //via global variable count
    count ← count + 1;   mark v with count and initialize a queue with v
    while the queue is not empty do
        for each vertex w in V adjacent to the front vertex do
            if w is marked with 0
                count ← count + 1;   mark w with count
                add w to the queue
        remove the front vertex from the queue
```


Notes on BFS

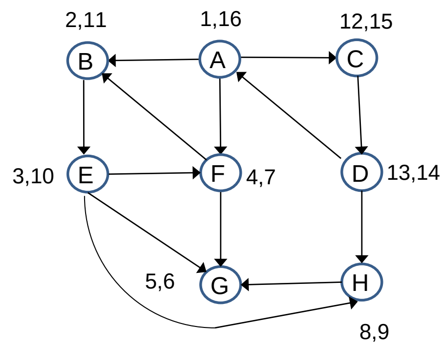
- BFS has same efficiency as DFS and can be implemented with graphs represented as:
  - adjacency matrices: $\Theta(V^2)$
  - adjacency lists: $\Theta(|V|+|E|)$
- Yields single ordering of vertices (order added/deleted from queue is the same)
- Applications: same as DFS, but can also find paths from a vertex to all other vertices with the smallest number of edges

DFS compared to BFS

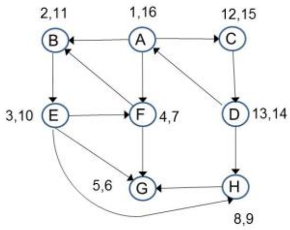
TABLE 3.1 Main facts about depth-first search (DFS) and breadth-first search (BFS)

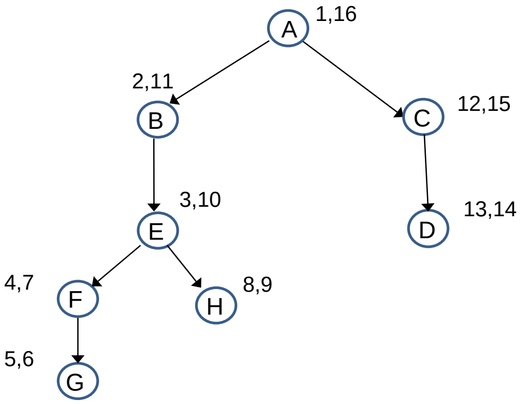
	DFS	**BFS**
Data structure	a stack	a queue
Number of vertex orderings	two orderings	one ordering
Edge types (undirected graphs)	tree and back edges	tree and cross edges
Applications	connectivity, acyclicity, articulation points	connectivity, acyclicity, minimum-edge paths
Efficiency for adjacency matrix	$\Theta(\lvert V^2 \rvert)$	$\Theta(\lvert V^2 \rvert)$
Efficiency for adjacency lists	$\Theta(\lvert V \rvert + \lvert E \rvert)$	$\Theta(\lvert V \rvert + \lvert E \rvert)$

Example: DFS traversal of directed graph



Example: DFS traversal of directed graph





Draw in the Back (2), Forward(2), and Cross Edges(2) – labeling appropriately.