Brute-force algorithm

Brute-force search or exhaustive search, also known as generate and test, is a very general problem-solving technique that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement.

Big O theta & big-omega

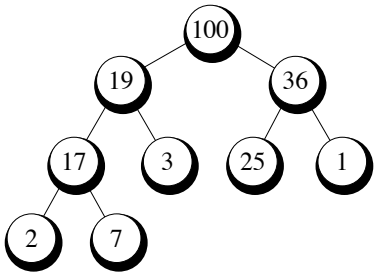
$f(x) = O(g(x))$ (big-oh)
means that the growth rate of f(x) is asymptotically ***less than or equal*** to to the growth rate of $g(x)$

$f(x) = \Omega(g(x))$ (big-omega)
means that the growth rate of f(x) is asymptotically ***greater than or equal*** to the growth rate of $g(x)$

$f(x) = \Theta(g(x))$ (theta)
means that the growth rate of f(x) is asymptotically ***equal*** to the growth rate of $g(x)$

heap (data structure)

A heap can be classified further as either a "max heap" or a "min heap". In a **max heap**, the keys of parent nodes are always greater than or equal to those of the children and the highest key is in the root node.



Matrix

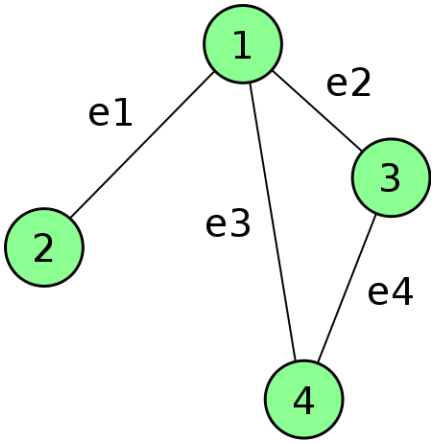
adjacency matrix

Space Complexity: $\Theta(V^2)$
an adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether **pairs of vertices are adjacent or not** in the graph.

$$\begin{pmatrix} 2 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

incidence matrix

Space Complexity: $\Theta(V * E)$
an incidence matrix is a matrix that shows the relationship between two classes of objects.



Recurrence relations

a subproblems of size n/b and then combining these answers in $O(n^d)$ time, for some $a, b, d > 0$ (in the multiplication algorithm, $a = 3$, $b = 2$, and $d = 1$). Their running time can therefore be captured by the equation $T(n) = aT([n/b]) + O(n^d)$.

Master theorem[2] *If* $T(n) = aT(\lceil n/b \rceil) + O(n^d)$ *for some constants* $a > 0$, $b > 1$, *and* $d \geq 0$, *then*

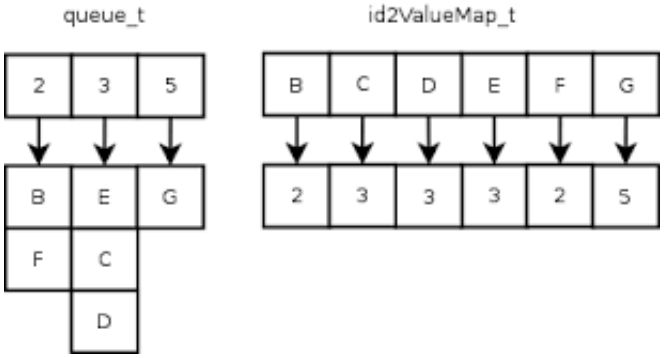
$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \,. \end{cases}$$

Binary search

The ultimate divide-and-conquer algorithm is, of course, *binary search:* to find a key k in a large file containing keys $z[0, 1, \ldots, n-1]$ in sorted order, we first compare k with $z[n/2]$, and depending on the result we recurse either on the first half of the file, $z[0, \ldots, n/2 - 1]$, or on the second half, $z[n/2, \ldots, n-1]$. The recurrence now is $T(n) = T(\lceil n/2 \rceil) + O(1)$, which is the case $a = 1, b = 2, d = 0$. Plugging into our master theorem we get the familiar solution: a running time of just $O(\log n)$.

Priortiy Queue

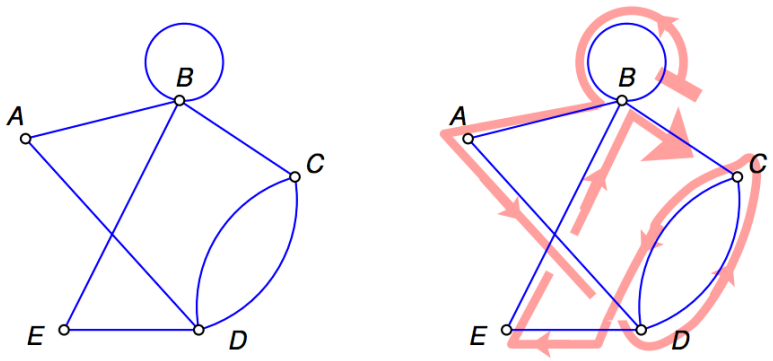
a priority queue is an abstract data type which is like a regular queue or stack data structure, but where **additionally each element has a "priority" associated with it**. In a priority queue, an element with high priority is served before an element with low priority. If two elements have the same priority, they are served according to their order in the queue.



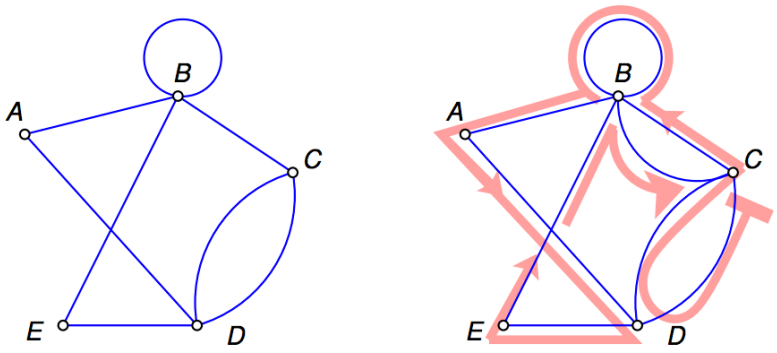
Euler Paths & Euler Circuit

An Euler path is a path that uses every edge of a graph exactly once.
An Euler circuit is a circuit that uses every edge of a graph exactly once.

An **Euler path** starts and ends at different vertices.
An **Euler circuit** starts and ends at the same vertex



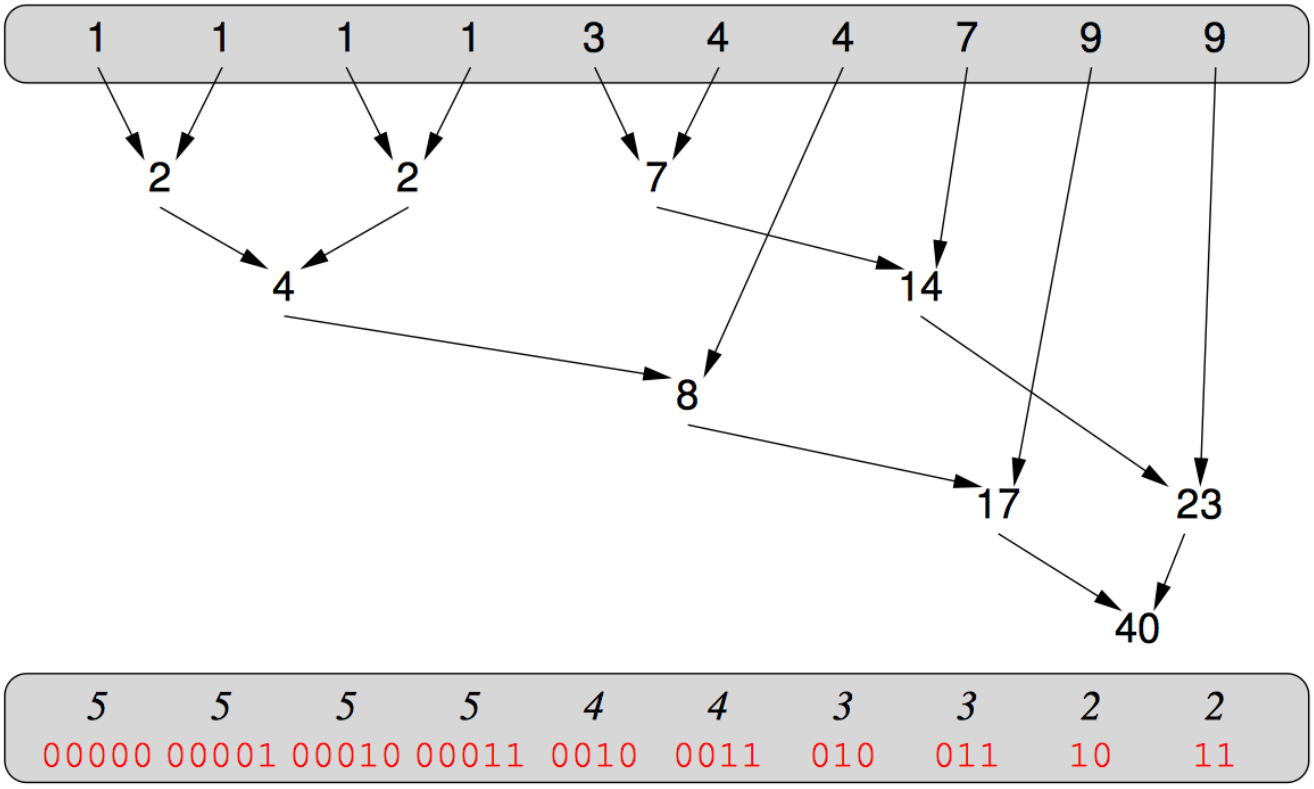
An Euler path: BBADCDEBC



An Euler circuit: CDCBBADEBC

Data Compression

Huffman encoding



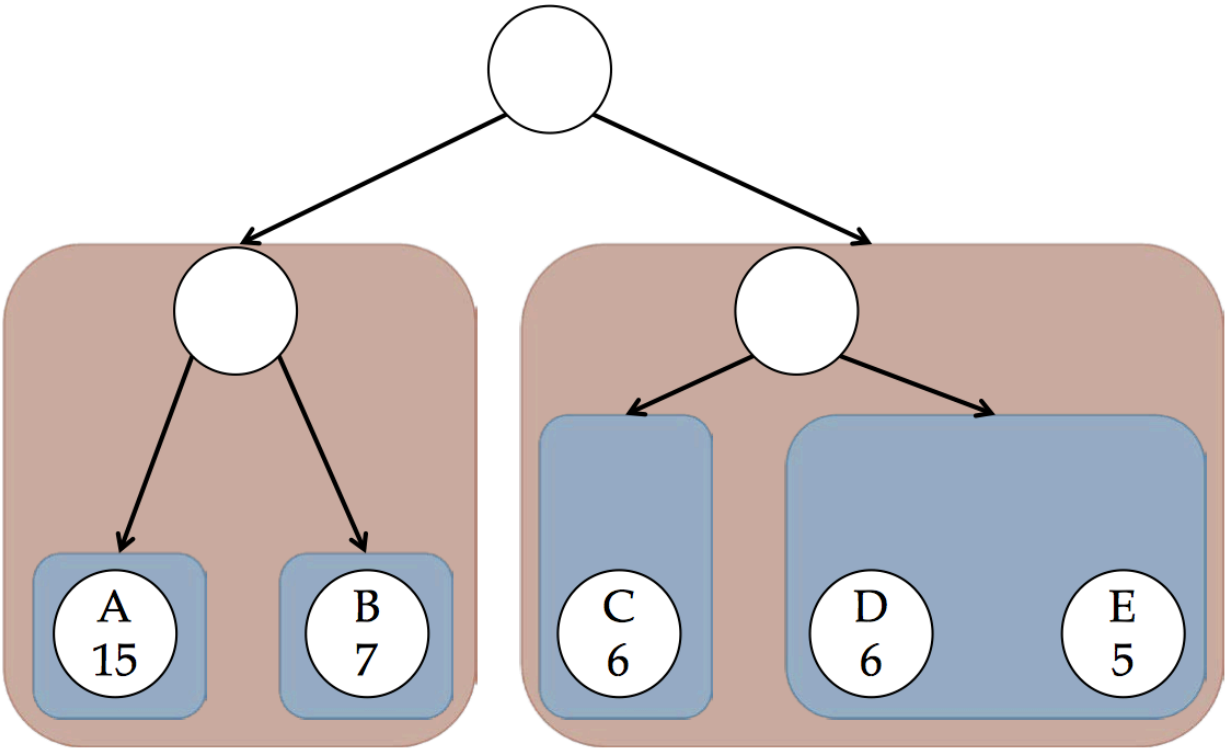
Cost in bits per symbol

- The data to be stored is called a **message**
- Each letter to be encoded is called a **symbol**
- Each bit pattern for each symbol is called a **codeword**
- A collection of **codewords** forms a **code**s
- We talk about “sending a message” and “storing some data” interchangeably. Storing data is just like sending it to disk.

- The cost of a code can be measured in bits-per-symbol, bps, for a particular message: total bits divided by the number of symbols
- ASCII always has the cost of 8 bps

$$bps = \frac{totalbits}{thenumberofsymbols}$$

Shannon-Fano coding



Shannon's entropy

If a symbol occurs with probability p, the best possible encoding for that symbol is

$$-log2p$$

Shannon’s entropy theorem:

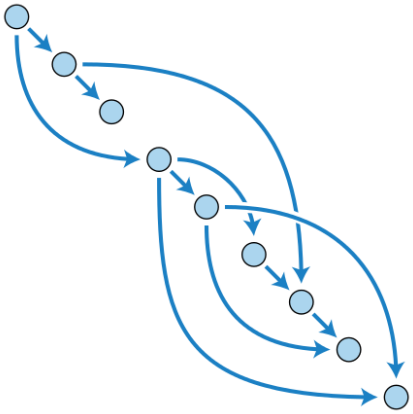
Given a message with n symbols each occurring p i times, the best bps possible is

$$H(p) = -\sum_{n}^{i=1} p_i log_2 p_i$$

Graph

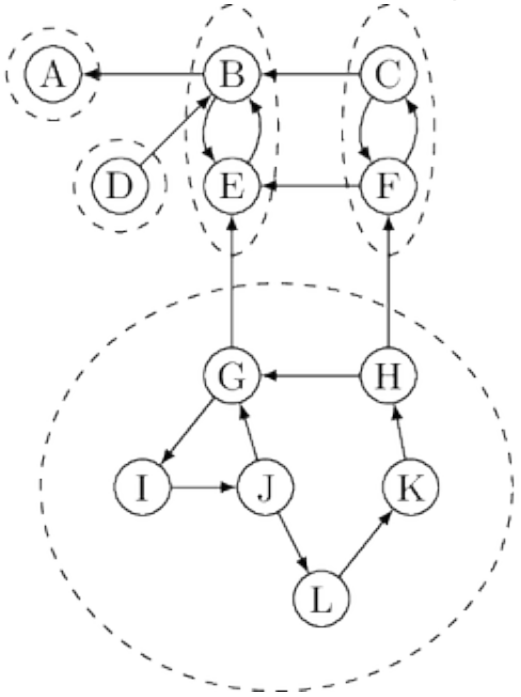
DAG (Directed acyclic graph)

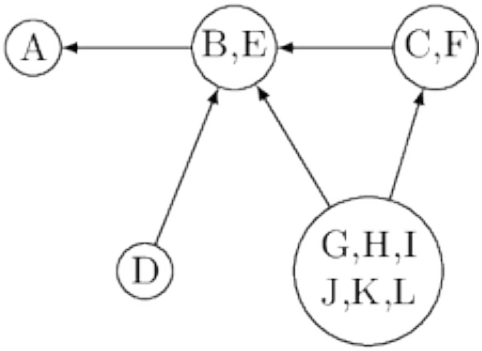
a finite directed graph with no directed cycles



SCC Strongly Connected components

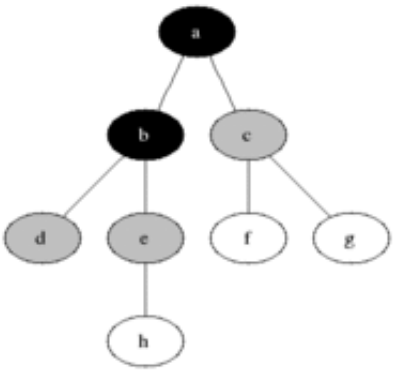
For each vertex in SCC, there is a path to any other vertex in the SCC





Source: a vertex with no incoming edges
Sink: a vertex with no outgoing edges

BFS Breath-fisrt Search



Complexity:

$$O(V+E)$$

DFS Depth-first Search

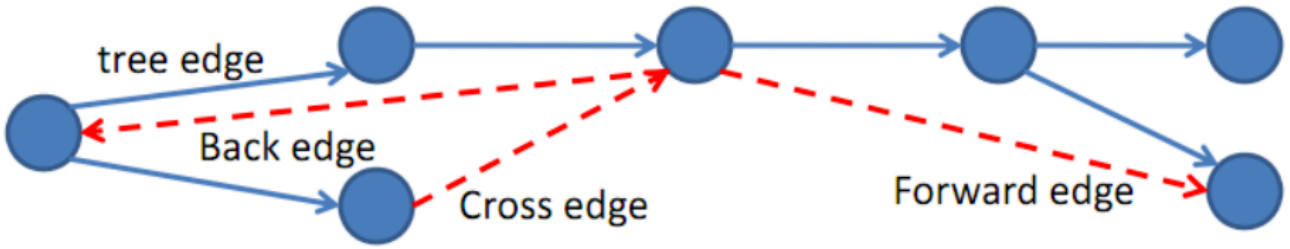
Complexity:

$$O(V+E)$$

Pseudo code

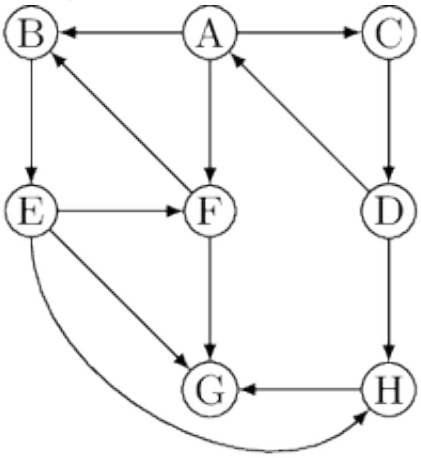
```
traverse(v):
    s = stack
    s.push(v)
    while s not empty:
        v = s.pop()
        mark v visited
        for each w adjacent to v:
            if w not visited:
                s.push(w)
```

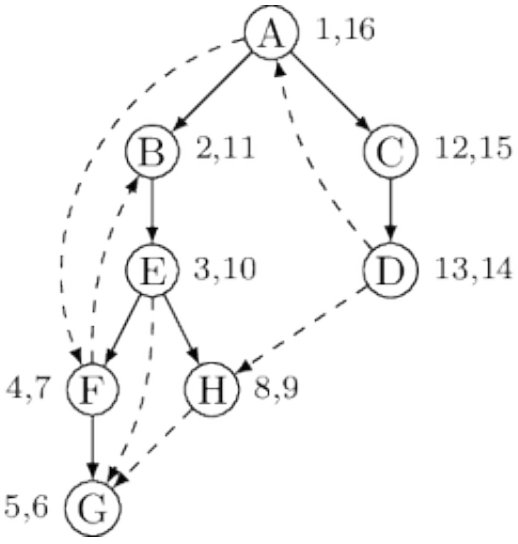
DFS Edge Classification



Pre and Post Number in DFS

Example

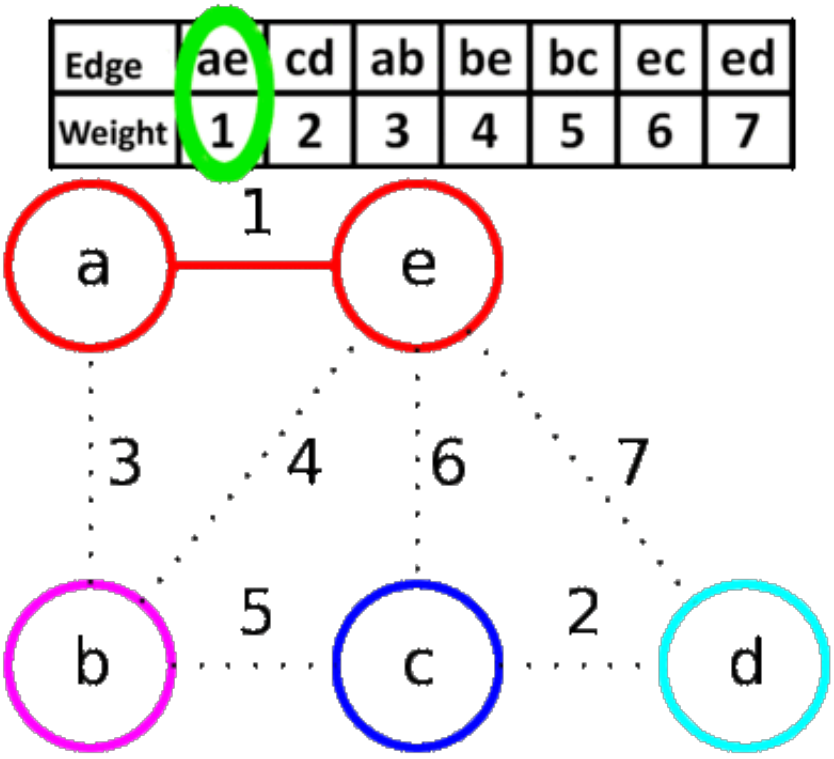




Greedy Algorithms Definition

A greedy algorithm is an algorithm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum.

Kruskal's Algorithm

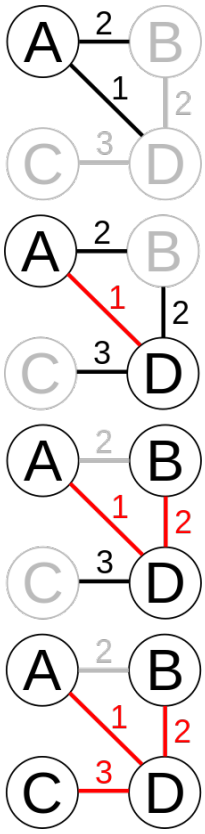



```
KRUSKAL(G):
A = ∅
```

```
foreach v ∈ G.V:
    MAKE-SET(v)
foreach (u, v) in G.E ordered by weight(u, v), increasing:
if FIND-SET(u) ≠ FIND-SET(
    A = A ∪ {(u, v)}
    UNION(u, v)
return A
```

Prim's Algorithm

Mimum spanning tree

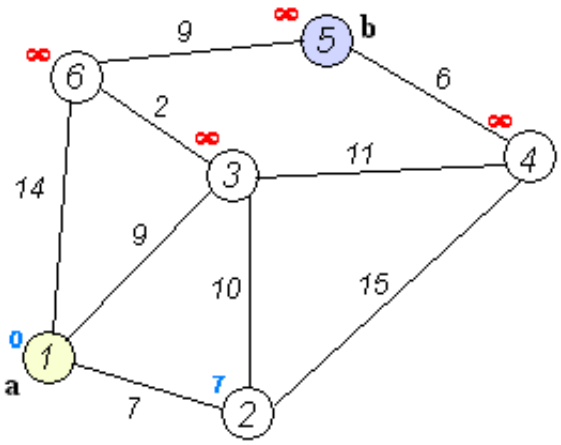


Complexity

Where **E** is the number of edges in the graph and **V** is the number of vertices, Kruskal's algorithm can be shown to run in **O(E log E)** time, or equivalently, **O(E log V)** time, all with simple data structures.

Dijkstra's Algorithm (find shortest path)

Complexity: O((V+E)log V)



Bellman-Ford algorithm

find shortest path when negative edge exists

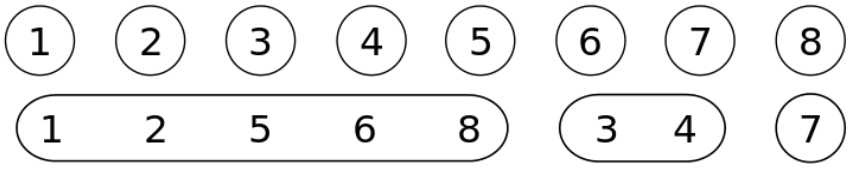
```
procedure shortest-paths(G, l, s)
Input:    Directed graph G = (V, E);
          edge lengths {l_e : e ∈ E} with no negative cycles;
          vertex s ∈ V
Output:   For all vertices u reachable from s, dist(u) is set
          to the distance from s to u.

for all u ∈ V:
   dist(u) = ∞
   prev(u) = nil

dist(s) = 0
repeat |V| − 1 times:
   for all e ∈ E:
      update(e)
```

Knapsack Problem

Disjoint Set

It is a data structure that keeps track of a set of elements partitioned into a number of disjoint (nonoverlapping) subsets.

- *Find*: Determine which subset a particular element is in.
- *Union*: Join two subsets into a single subset.



Complexity

- makeset $O(1)$
- find worst $O(V)$
- union $O(1)$

Kahn's Algorithm

```
function toposort_kahn

    input: V and E in graph G (a dag)
    output: x - the topological sorted vertices

    q = a queue with all vertices having no incoming edge // O(V)
    x = an empty linked list

    while q is not empty

        v = q.dequeue() // O(1) * O(V) times
        x.append(v) // O(1) * O(V) times

        for e, u in outgoing edges of v such that e(v,u)
            delete e // O(1) * O(E) times
            if u doesn't have any incoming edge // O(1) * O(E) times
                q.enqueue(u) // O(1) * O(E) times

    via Xiao Liang Yu
```

Complexity

Time Complexity: $O(V + E)$

Comparison Sort

A **comparison** sort is a type of sorting algorithm that only reads the list elements through a single abstract comparison operation (often a "less than or equal to" operator or a three-way comparison) that determines which of two elements should occur first in the final sorted list.

Hashing

Secret Sharing

Dynamic Programming

RSA

c = encrypted data
d = private key
n = public key
Decrypted Data = c^d mod n

Number Theory

Modular arithmetic

(A $*$ B) mod C = (A mod C $*$ B mod C) mod C

(A + B) mod C = (A mod C + B mod C) mod C

A^B mod C = ((A mod C)^B mod C

NP/P/NP-Completeness

K-th smallest algorithm