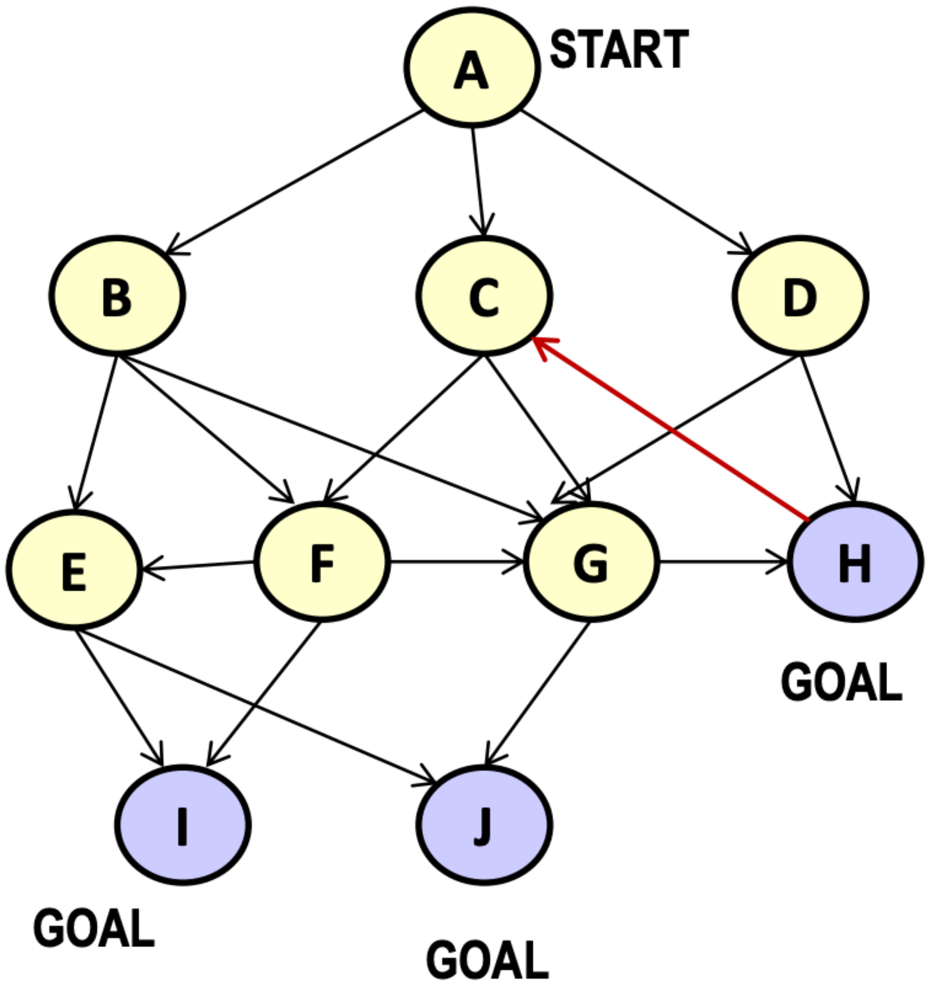
ARTIFICIAL INTELLIGENCE (AI)
STATE SPACE AND SEARCH ALGORITHMS

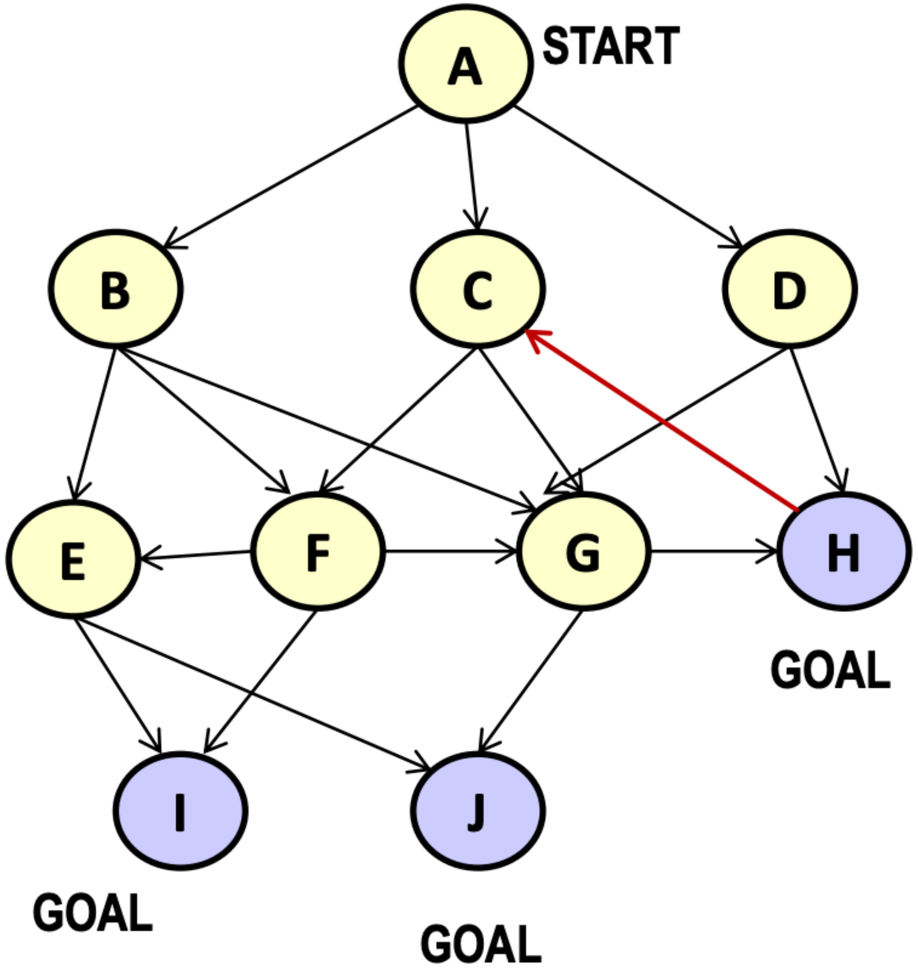
Shreya Ghosh
Assistant Professor, Computer Science and Engineering
IIT Bhubaneswar

EXAMPLE: SEARCHING A STATE SPACE GRAPH



- DEPTH-FIRST SEARCH (DFS)
- BREADTH-FIRST SEARCH (BFS)
- ITERATIVE DEEPENDING SEARCH (IDS)
- PROPERTIES
  - SOLUTION GUARANTEES
  - MEMORY REQUIREMENTS

EXAMPLE: SEARCHING A STATE SPACE GRAPH



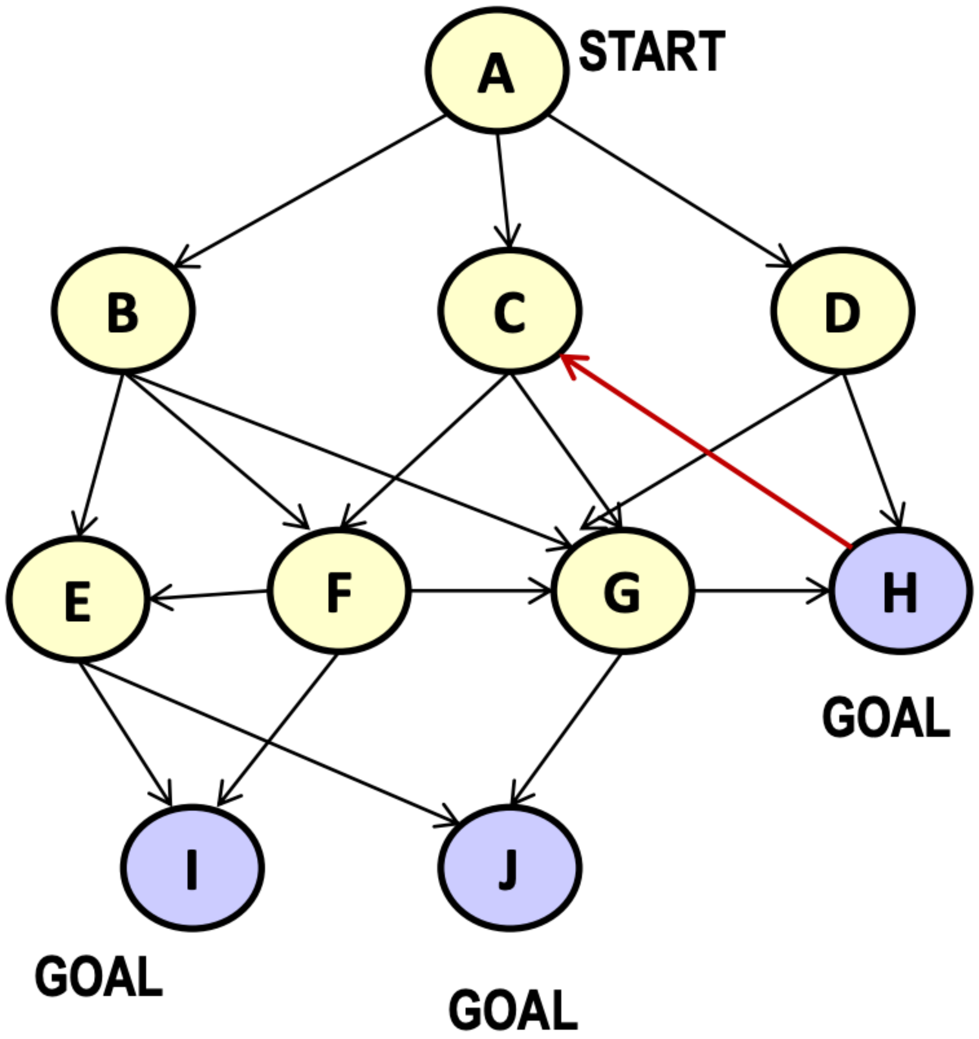
DEPTH-FIRST SEARCH:

1. OPEN ={A}, CLOSED = {}
2. OPEN = {B,C,D}, CLOSED = {A}
3. OPEN = {E,F,G,C,D}, CLOSED = (A,B}
4. OPEN = {I,J,F,G,C,D}, CLOSED = {A,B,E}
5. Goal Node I Found. Can Terminate with Path from A to I or may Continue for more Goal nodes if minimum length or cost is a criteria

DFS MAY NOT TERMINATE IF THERE IS AN INFINITE DEPTH PATH EVEN IF THERE IS A GOAL NODE AT FINITE DEPTH

DFS HAS LOW MEMORY REQUIREMENT

EXAMPLE: SEARCHING A STATE SPACE GRAPH



ITERATIVE DEEPENING SEARCH:

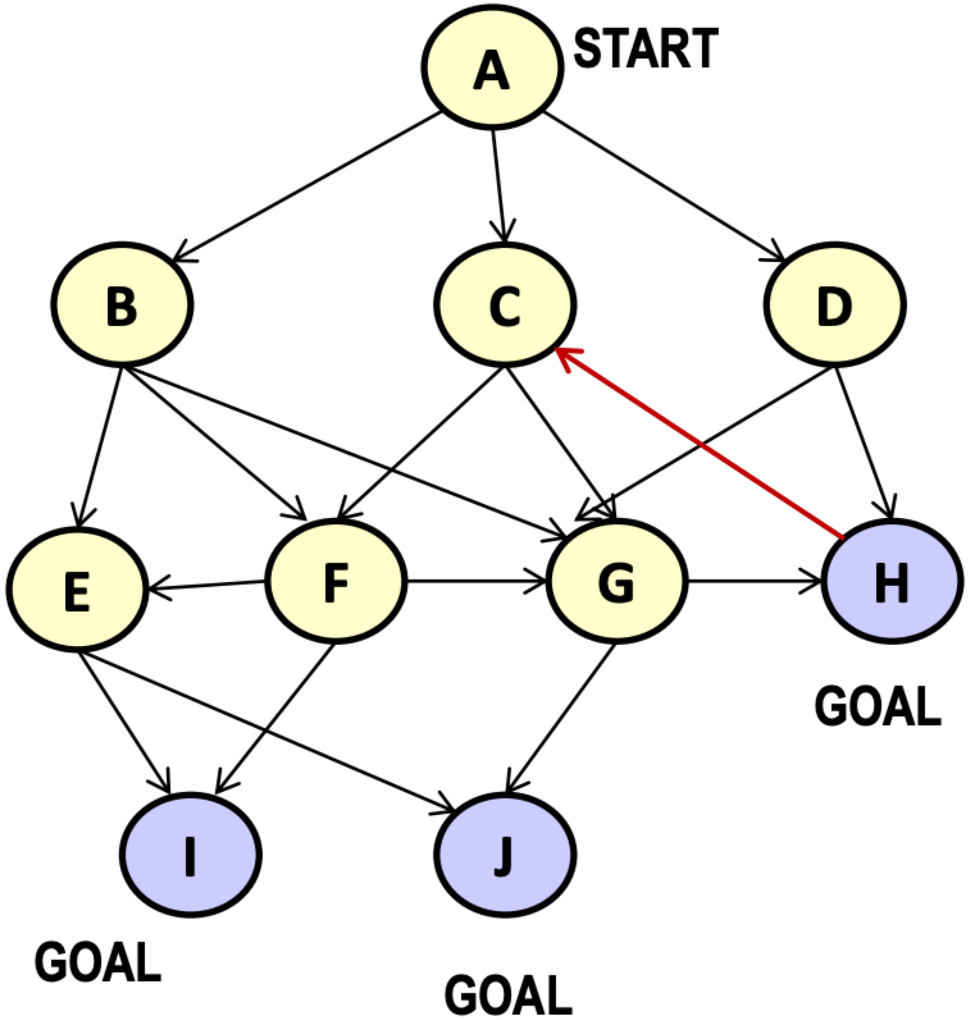
1. PERFORM DFS TILL LENGTH 1. NO SOLUTION FOUND
2. PERFORM DFS TILL LEVEL 2. GOAL NODE H REACHED.
3. Can Terminate with Path from A to H. This is guaranteed to be the minimum length path.

IDS GUARANTEES SHORTEST LENGTH PATH TO GOAL

IDS MAY RE-EXPAND NODES MANY TIMES

IDS HAS LOWER MEMORY REQUIREMENT THAN BFS

EXAMPLE: SEARCHING A STATE SPACE GRAPH

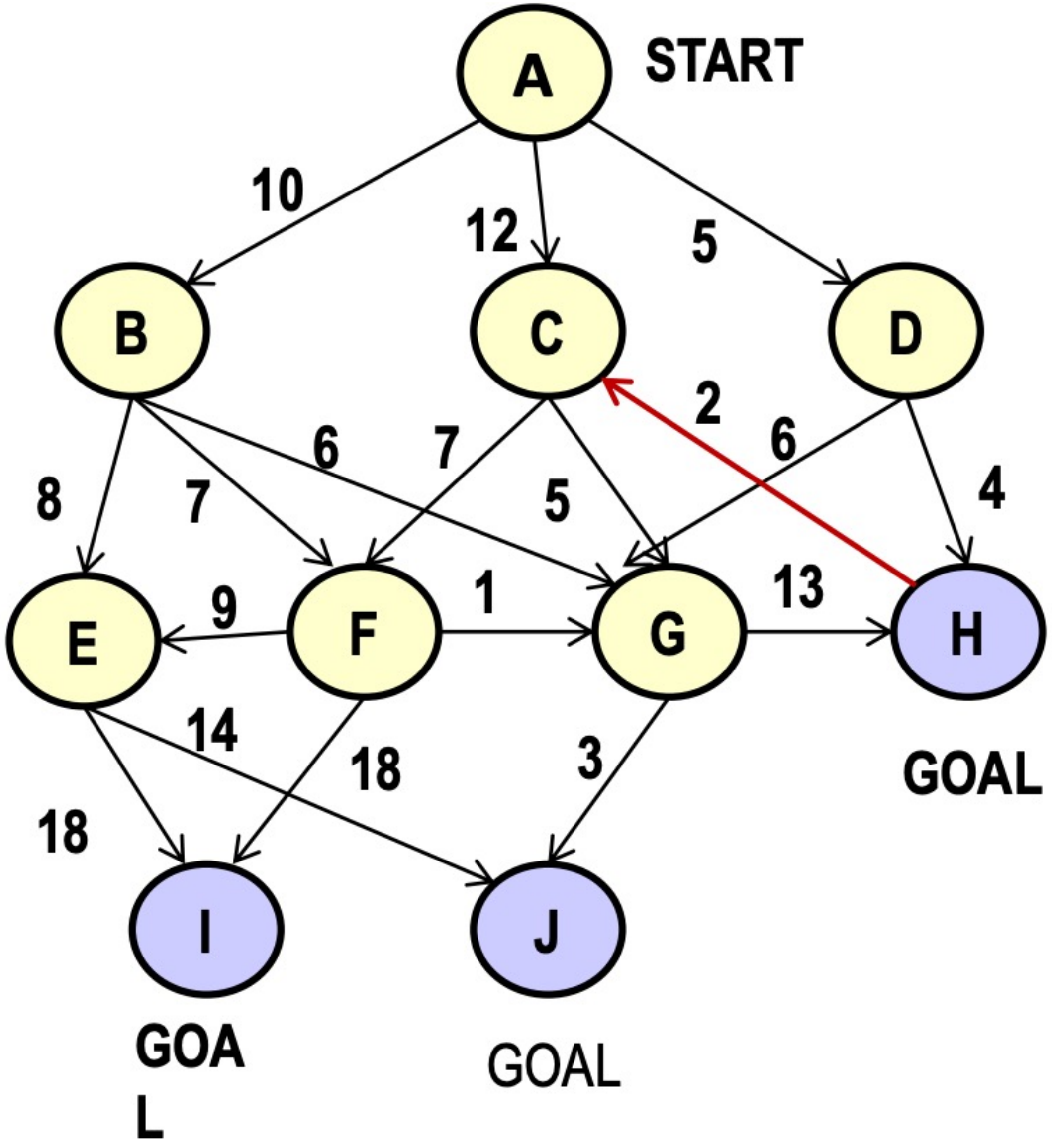


BREADTH-FIRST SEARCH:

1. OPEN ={A}, CLOSED = {}
2. OPEN = {B,C,D}, CLOSED = {A}
3. OPEN = {C,D,E,F,G}, CLOSED = (A,B}
4. OPEN = {D,E,F,G}, CLOSED = {A,B,C}
5. OPEN = {E,F,G,H}. CLOSED = {A,B,C,D}
6. OPEN = {F,G,H,I,J}, CLOSED = {A,B,C,D,E}
7. OPEN = {G,H,I,J}, CLOSED = {A,B,C,D,E,F}
8. OPEN = {H,I,J}, CLOSED = {A,B,C,D,E,F,G}
9. Goal Node H Found. Can Terminate with Path from A to H. This is guaranteed to be the minimum length path.

BFS GUARANTEES SHORTEST LENGTH PATH TO GOAL BUT HAS HIGHER MEMORY REQUIREMENT

SEARCHING STATE SPACE GRAPHS WITH EDGE COSTS

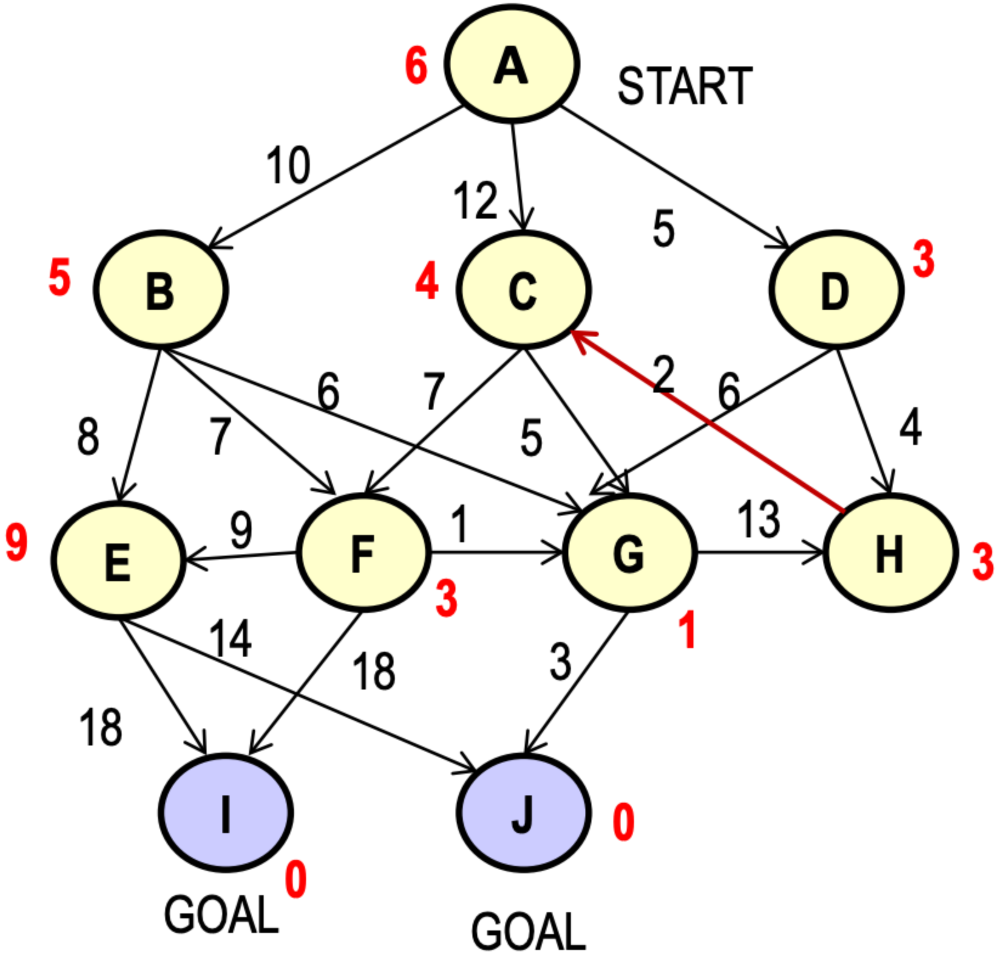


ALGORITHM A* (BEST FIRST SEARCH IN OR GRAPHS)

Each Node n in the algorithm has a cost g(n) and a heuristic estimate h(n), f(n) = g(n + h(n).
Assume all c(n,m) > 0

1. [Initialize] Initially the OPEN List contains the Start Node s. g(s) = 0, f(s) = h(s). CLOSED List is Empty.
2. [Select] Select the Node n on the OPEN List with minimum f(n). If OPEN is empty, Terminate with Failure
3. [Goal Test, Terminate] If n is Goal, then Terminate with Success and path from s to n.
4. [Expand]
   a) Generate the successors n_1, n_2, …. n_k, of node n, based on the State Transformation Rules
   b) Put n in LIST CLOSED
   c) For each n_i, not already in OPEN or CLOSED List, compute
      a) g(n_i) = g(n) + c(n, n_i), f(n_i) = g(n_i) + h(n_i) , Put n_i in the OPEN List
   d) For each n_i already in OPEN, if g(n_i) > g(n) + c(n,n_i), then revise costs as:
      a) g(n_i) = g(n) + c(n, n_i), f(n_i) = g(n_i) + h(n_i)
5. [Continue] Go to Step 2

PROPERTIES OF ALGORITHM A*



IF HEURISTIC ESTIMATES ARE NON-NEGATIVE LOWER BOUNDS AND EDGE COSTS ARE POSITIVE:

- FIRST SOLUTION IS OPTIMAL
- NO NODE IN CLOSED IN EVER REOPENED
- WHENEVER A NODE IS REMOVED FROM OPEN ITS MINIMUM COST FROM START IS FOUND
- EVERY NODE n WITH f(n) LESS THAN OPTIMAL COST IS EXPANDED
- IF HEURISTICS ARE MORE ACCURATE THEN SEARCH IS LESS

ALGORITHM DFBB

DEPTH FIRST BRANCH AND BOUND (DFBB)

1. Initialize Best-Cost to INFINITY
2. Perform DFS with costs and Backtrack from any node n whose $f(n) \geq$ Best-Cost
3. On reaching a Goal Node, update Best-Cost to the current best
4. Continue till OPEN becomes empty

ALGORITHM IDA*

ITERATIVE DEEPENING A* (IDA*)

1. Set Cut-off Bound to f(s)
2. Perform DFBB with Cut-off Bound. Backtrack from any node whose f(n) > Cut-off Bound.
3. If Solution is Found, at the end of one Iteration, Terminate with Solution
4. If Solution is not found in any iteration, then update Cut-off Bound to the lowest f(n) among all nodes from which the algorithm Backtracked.
5. Go to Step 2
6. PROPERTIES OF DFBB AND IDA*: Solution Cost, Memory, Node expansions, Heuristic Accuracy, Performance on Trees / Graphs

KNAPSACK

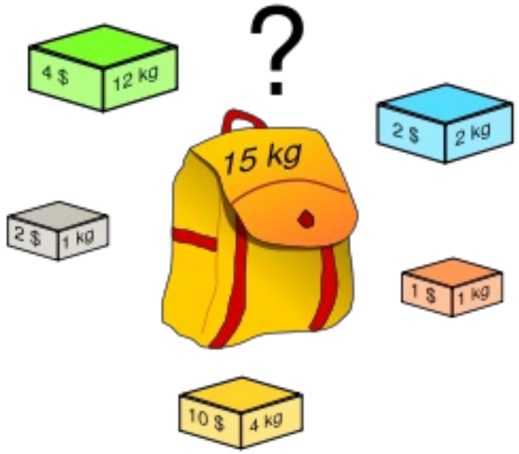
- The **knapsack problem** or **rucksack problem** is a problem in combinatorial optimization.

- It derives its name from the following maximization problem of the best choice of essentials that can fit into one bag to be carried on a trip.

- Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than a given limit and the total value is as large as possible.

THE ORIGINAL KNAPSACK PROBLEM (1)

- Problem Definition
  - Want to carry essential items in one bag
  - Given a set of items, each has
    - A cost (i.e., 12kg)
    - A value (i.e., 4$)



- Goal
  - To determine the # of each item to include in a collection so that
    - The total cost is less than some given cost
    - And the total value is as large as possible

THE ORIGINAL KNAPSACK PROBLEM (2)

- Three Types
  - 0/1 Knapsack Problem
    - restricts the number of each kind of item to zero or one
  - Bounded Knapsack Problem
    - restricts the number of each item to a specific value
  - Unbounded Knapsack Problem
    - places no bounds on the number of each item

- Complexity Analysis
  - The general knapsack problem is known to be NP-hard
    - No polynomial-time algorithm is known for this problem
  - Here, we use greedy heuristics which cannot guarantee the optimal solution