Backtracking and Branch-and-Bound Techniques

Mr. Channa Bankapur
channabankapur@pes.edu

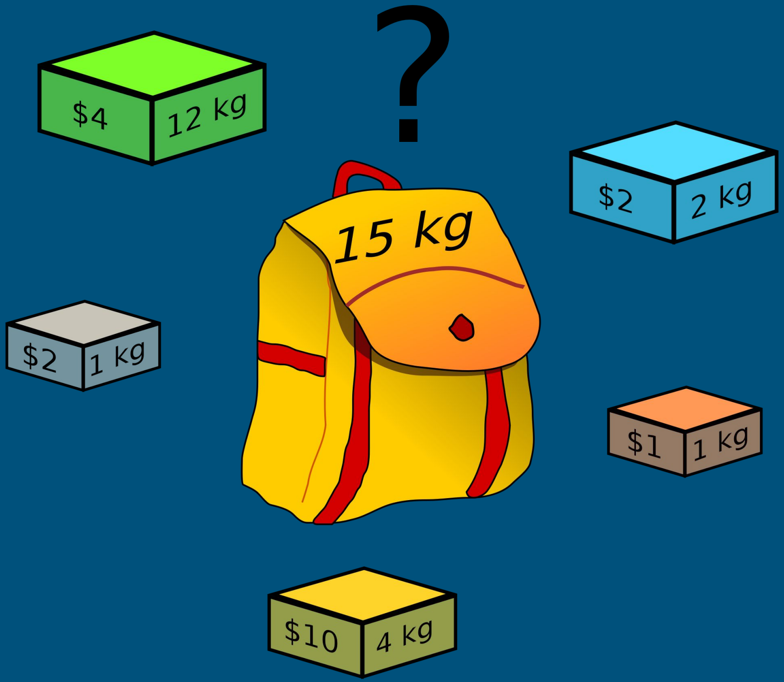
0/1 Knapsack Problem:

Given n items of (*value*,*weight*) pairs

items$[(v_0,w_0)\ldots(v_{n-1},w_{n-1})]$

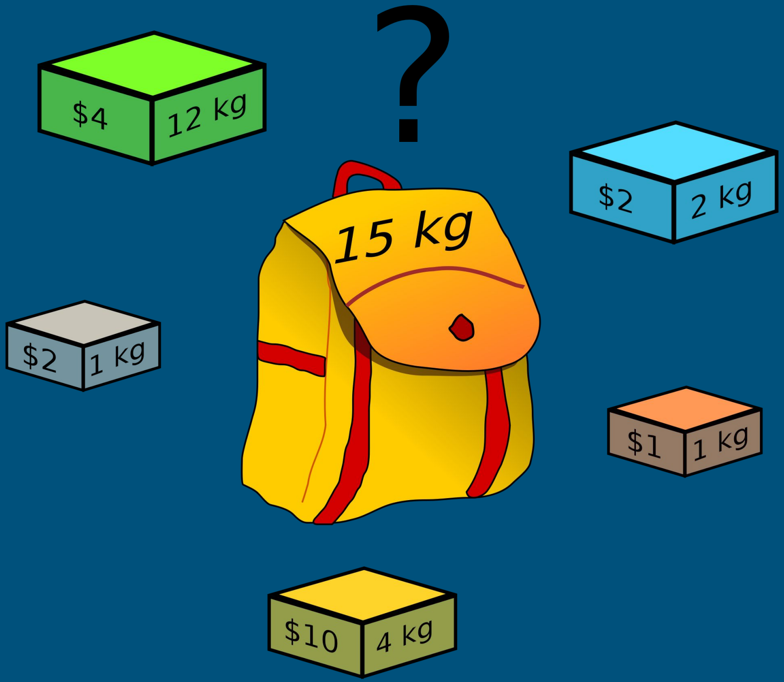
knapsack capacity: C

Choose as many items as possible that fit into the knapsack by weight while maximizing the overall value.



Greedy method:

```
Knapsack(items[0..n-1], C)
   //Sort by higher value per unit weight
   sort(items)
   val = 0
   for i=0 to n-1
      if(C >= items[i].wt)
         val = val + items[i].val
         C = C - items[i].wt
   return val
```



Eg: Knapsack values:

\$10 + \$2 + \$2 + \$1 = \$15

4kg + 1kg + 2kg + 1kg = 8kg

Eg: items by (value,weight) pairs
Sorted items: (\$8, 2kg), (\$15, 5kg), (\$6, 3kg), (\$2, 2kg)
Capacity C = 6 kg

Greedy Method:
\$8 + \$6 = \$14

Optimal value:
\$15

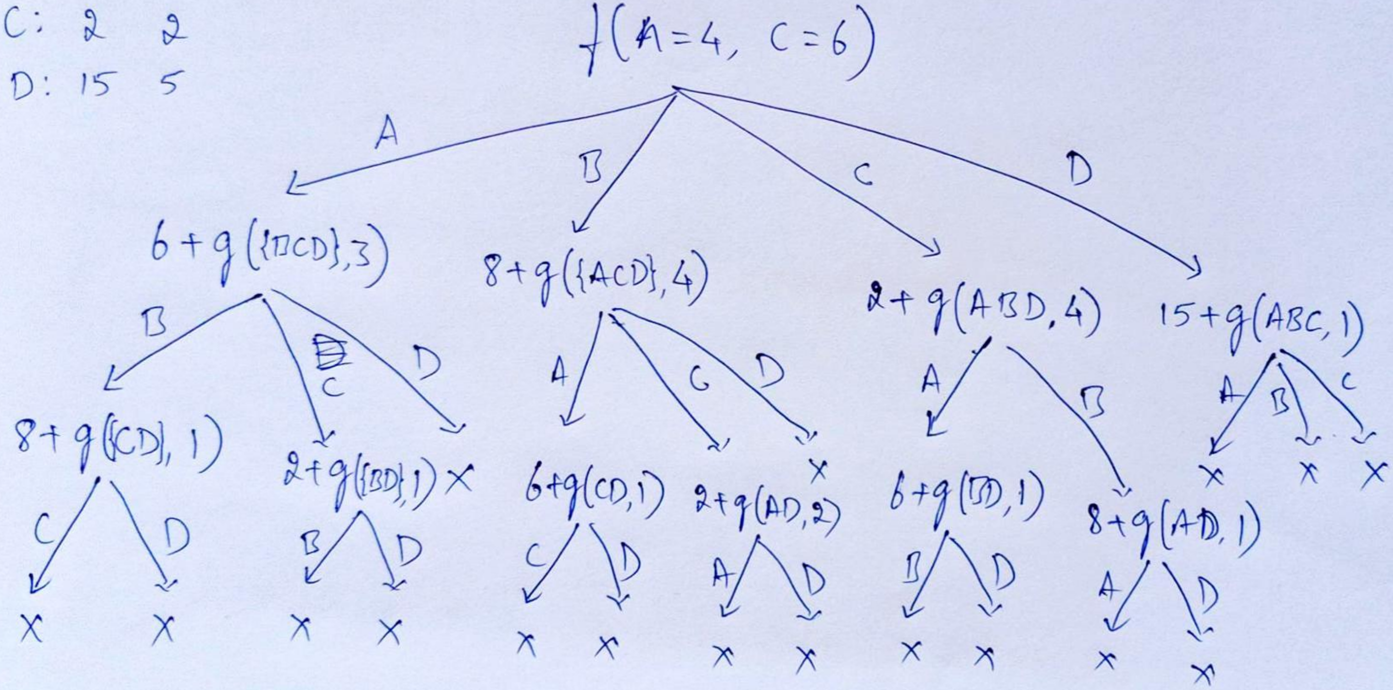
	v_i	w_i
A:	6	3
B:	8	2
C:	2	2
D:	15	5

$f(n=4, C=6)$

	v_i	w_i
A:	6	3
B:	8	2
C:	2	2
D:	15	5

$$\begin{tikzcd} & & f(A=4,\ C=6) \arrow[dll, "A"'] \arrow[dl, "B"] \arrow[dr, "C"] \arrow[drr, "D"] & & \\ 6+g(\{BCD\},3) & 8+g(\{ACD\},4) & & 2+g(ABD,4) & 15+g(ABC,1) \end{tikzcd}$$

	v_i	w_i
A:	6	3
B:	8	2
C:	2	2
D:	15	5



Exhaustive Search method:

```
Knapsack(items[0..n-1], C)
   val = 0
   for each subset of items
      if(weight(subset) <= C)
         if(value(subset) > val)
            val = value(subset)
   return val
```

$T(n) = O(\mathbf{n\ 2^n})$

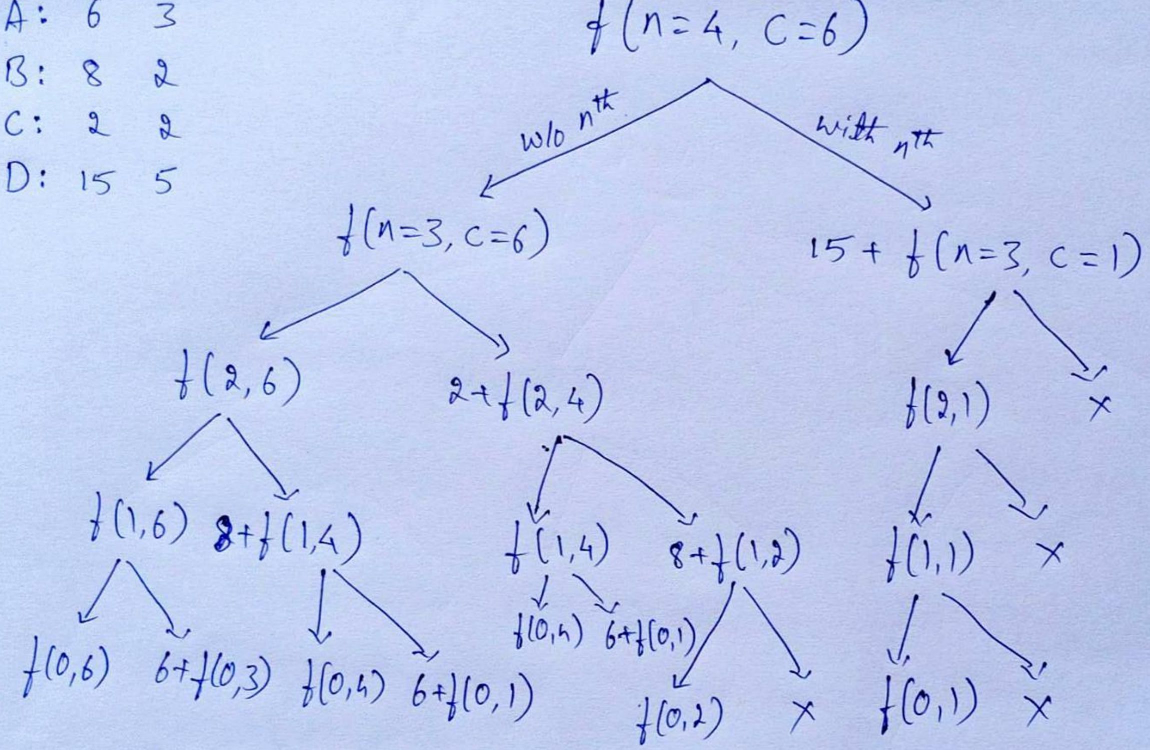
Eg: items by (value,weight) pairs
($6, 3kg), (8, 2), (2, 2), (15, 5)
Capacity C = 6 kg

0000: $0
0001: $6
0010: $8
0011: $14
0100: $2
0101: $8
0110: $10
0111: Infeasible
1000: $15
1001: Infeasible
1010: Infeasible
1011: Infeasible
1100: Infeasible
1101: Infeasible
1110: Infeasible
1111: Infeasible

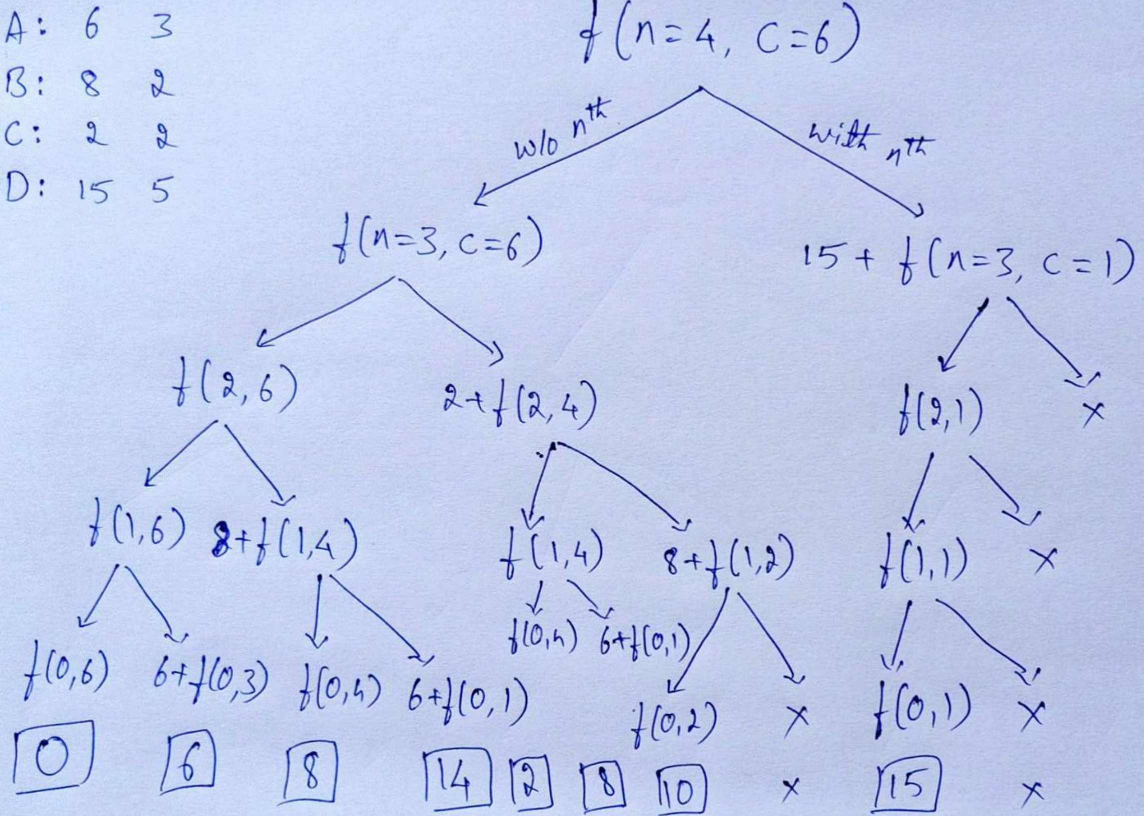
	v_i	w_i
A:	6	3
B:	8	2
C:	2	2
D:	15	5

$$
\begin{tikzcd}[column sep=small]
 & & & f(n=4,\ C=6) \arrow[dl, "w/o\ n^{th}"'] \arrow[drr, "with\ n^{th}"] & & \\
 & & f(n=3,\ C=6) \arrow[dl] \arrow[dr] & & & 15+f(n=3,\ C=1) \\
 & f(2,6) \arrow[dl] \arrow[dr] & & 2+f(2,4) & & \\
f(1,6) \arrow[d] \arrow[dr] & & 8+f(1,4) \arrow[d] \arrow[dr] & & & \\
f(0,6) & 6+f(0,3) & f(0,4) & 6+f(0,1) & &
\end{tikzcd}
$$

	v_i	w_i
A:	6	3
B:	8	2
C:	2	2
D:	15	5



	v_i	w_i
A:	6	3
B:	8	2
C:	2	2
D:	15	5



$$\begin{tikzcd}[column sep=small] & f(n, c) \arrow[dl, "\text{without } n^{th}"'] \arrow[dr, "\text{with } n^{th} \text{ if } w_n \le c"] \arrow[d, phantom, "\max"] & \\ f(n-1, c) & \; & v_n + f(n-1, c-w_n) \end{tikzcd}$$

$$f(n, c) = \begin{cases} f(n-1, c) & \text{if } w_n > c \\ \max\{f(n-1, c),\ v_n + f(n-1, c-w_n)\} & \end{cases}$$

$$f(0, c) = 0$$

$$f(n, 0) = 0$$

Backtracking
The exhaustive search technique suggests generating all candidate solutions and then identifying the one (or the ones) with a desired property.

- Construct solutions one component at a time and evaluate such partially constructed candidates.
- Construct the **state-space tree**
  - **non-leaf nodes**: promising nodes with partial solutions
  - **leaves**: non-promising nodes or solutions
  - **edges**: choices in extending partial solutions
- Explore the state space tree using **depth-first search**.
- **Backtrack** at non-promising nodes.

	v_i	w_i	v_i/w_i
A	6	3	2
B	8	2	4
C	2	2	1
D	15	5	3

	v_i	w_i	v_i/w_i
B:	8	2	4
D:	15	5	3
A:	6	3	2
C:	2	2	1

$f(n=4,\ C=6,\ ub=6*4=24,\ v=0)$

$\xrightarrow{\text{with } n^{th}\ (B)}$

$f(n=3,\ C=4,\ v=8,\ ub=8+12=20)$

$\xrightarrow{(D)}$ ×

$\xrightarrow{w/o\ (D)}$ $f(2, 4, 8, ub=16)$

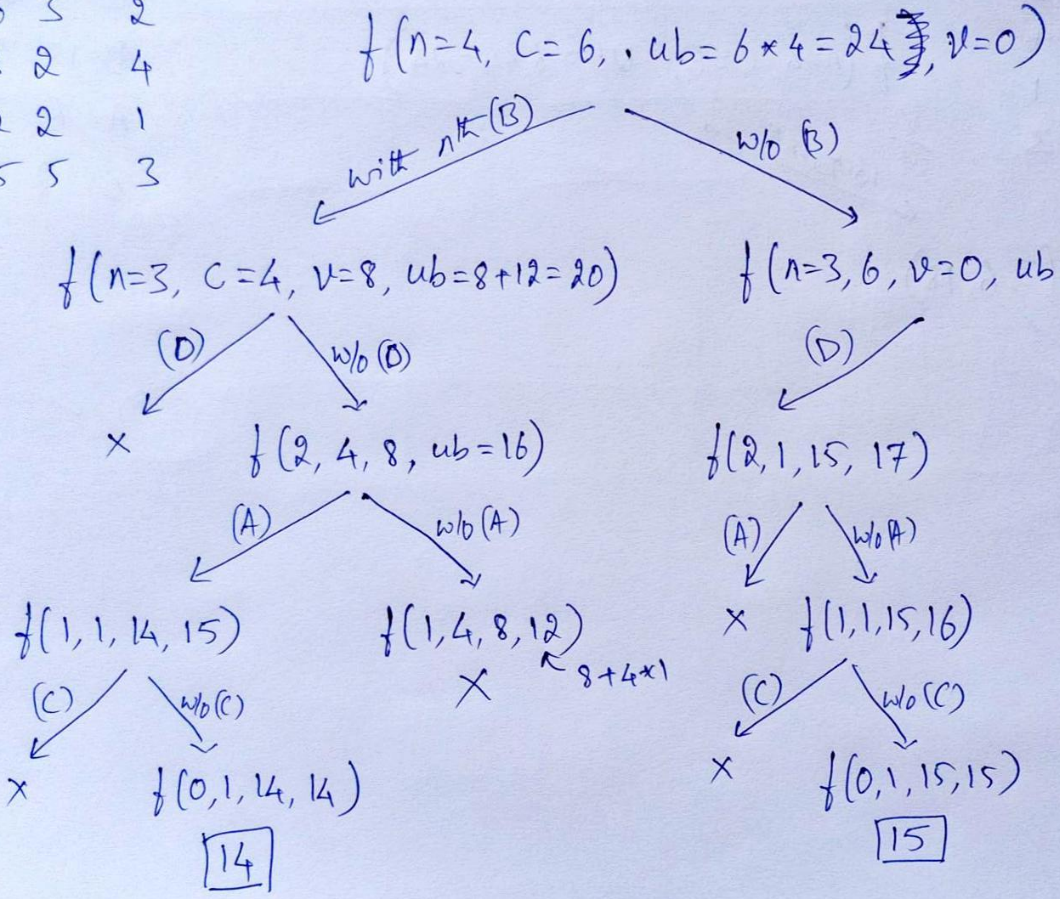
$\xrightarrow{(A)}$ $f(1, 1, 14, 15)$

$\xrightarrow{(C)}$ ×

$\xrightarrow{w/o\ (C)}$ $f(0, 1, 14, 14)$

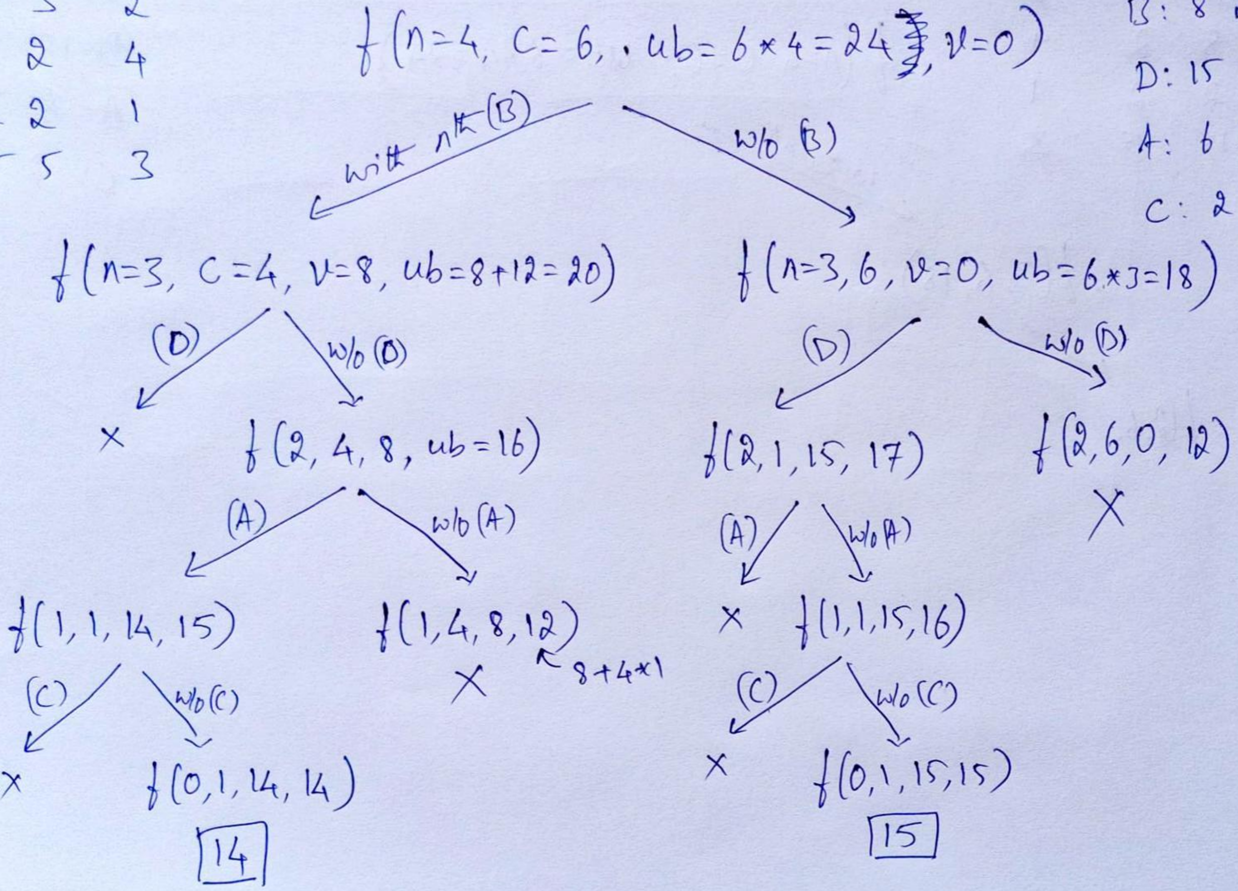
$\boxed{14}$

	v_i	w_i	v_i/w_i
A	6	3	2
B	8	2	4
C	2	2	1
D	15	5	3



	v_i	w_i	v_i/w_i
B:	8	2	4
D:	15	5	3
A:	6	3	2
C:	2	2	1

	v_i	w_i	v_i/w_i
A	6	3	2
B	8	2	4
C	2	2	1
D	15	5	3



	v_i	w_i	v_i/w_i
B:	8	2	4
D:	15	5	3
A:	6	3	2
C:	2	2	1

Branch-and-Bound

- An enhancement of backtracking
- Applicable to optimization problems
- Makes a note of the best solution seen so far
- For each node (partial solution) of a state-space tree, computes a **bound** on the value of the objective function for all descendants of the node (extensions of the partial solution)

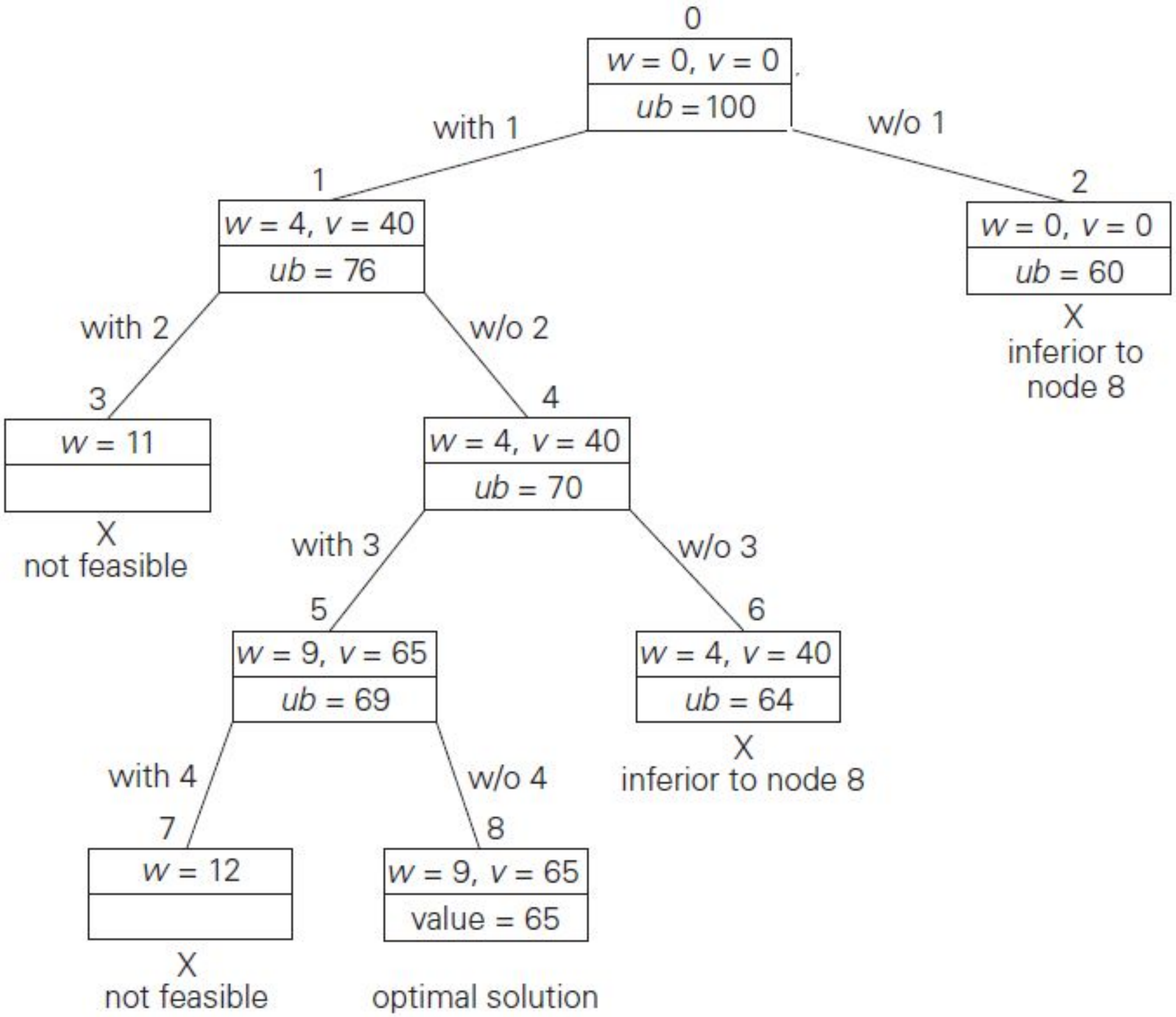
Branch-and-Bound - Knapsack Problem

$$v_1/w_1 \geq v_2/w_2 \geq \cdots \geq v_n/w_n.$$

$$ub = v + (W - w)(v_{i+1}/w_{i+1}).$$

item	weight	value	$\frac{\text{value}}{\text{weight}}$
1	4	$40	10
2	7	$42	6
3	5	$25	5
4	3	$12	4

The knapsack's capacity W is 10.



Travelling Salesperson Problem

1. Bengaluru
2. New Delhi
3. Mumbai
4. Chennai
5. Kolkata
6. Kochi
7. Hyderabad
8. Bhopal
9. Udaipur
10. Raipur

TSP is an optimization problem

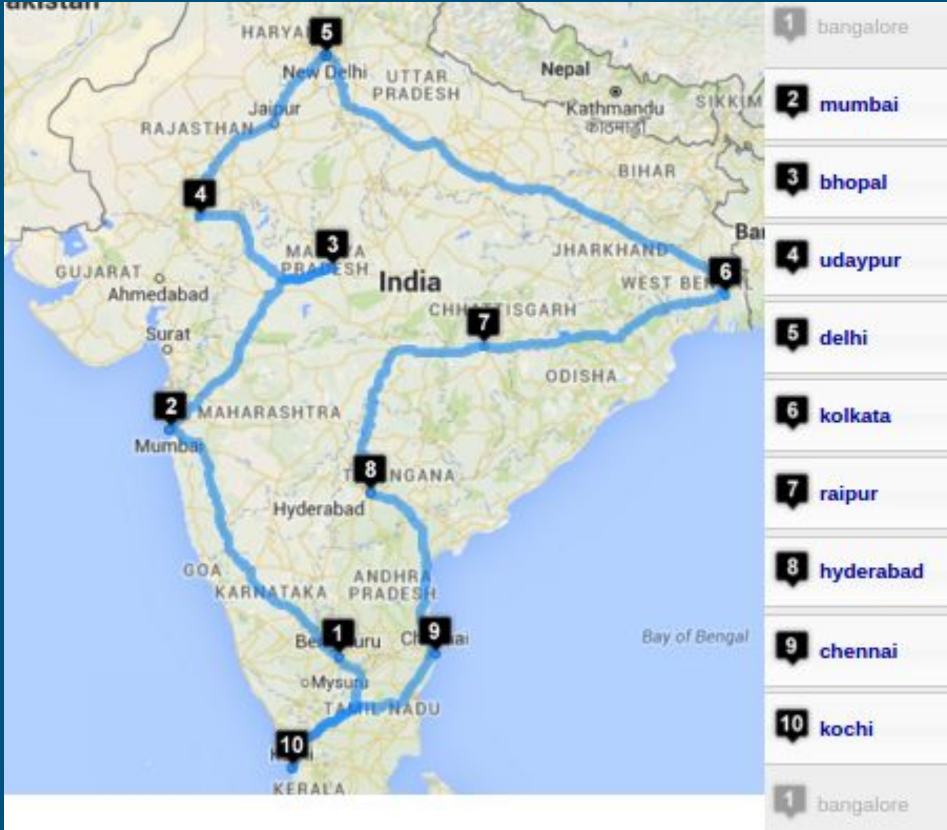
Estimated driving time (in seconds) from a city to another

	Blr	Delhi	Mumb	Chen	Kol	Kochi	Hyd	Bhopal	Udai	Raipur
Bengaluru	000000	110189	050573	020948	109480	034435	028433	074836	091767	068406
New Delhi	109006	000000	079663	118195	079397	143304	083593	045792	037923	068146
Mumbai	051516	080265	000000	070149	121881	083636	044745	043763	042416	067450
Chennai	021557	119539	069838	000000	095820	042397	037471	084186	111032	077756
Kolkata	110053	081231	121373	095977	000000	134475	085826	087690	100264	054016
Kochi	034488	144238	082769	041728	134042	000000	062482	108885	123963	102455
Hyderabad	028473	084770	045153	037117	085732	062772	000000	049417	078006	042987
Bhopal	075056	046162	044536	084245	086579	109354	049641	000000	031151	038399
Udaipur	092933	037994	042414	111566	099497	125053	078960	031010	000000	068113
Raipur	068718	068844	068336	077907	055357	103016	043305	038648	068634	000000

Traveling Salesperson Problem

1. Bengaluru
2. Mumbai
3. Bhopal
4. Udaipur
5. New Delhi
6. Kolkata
7. Raipur
8. Hyderabad
9. Chennai
10. Kochi
11. Bengaluru

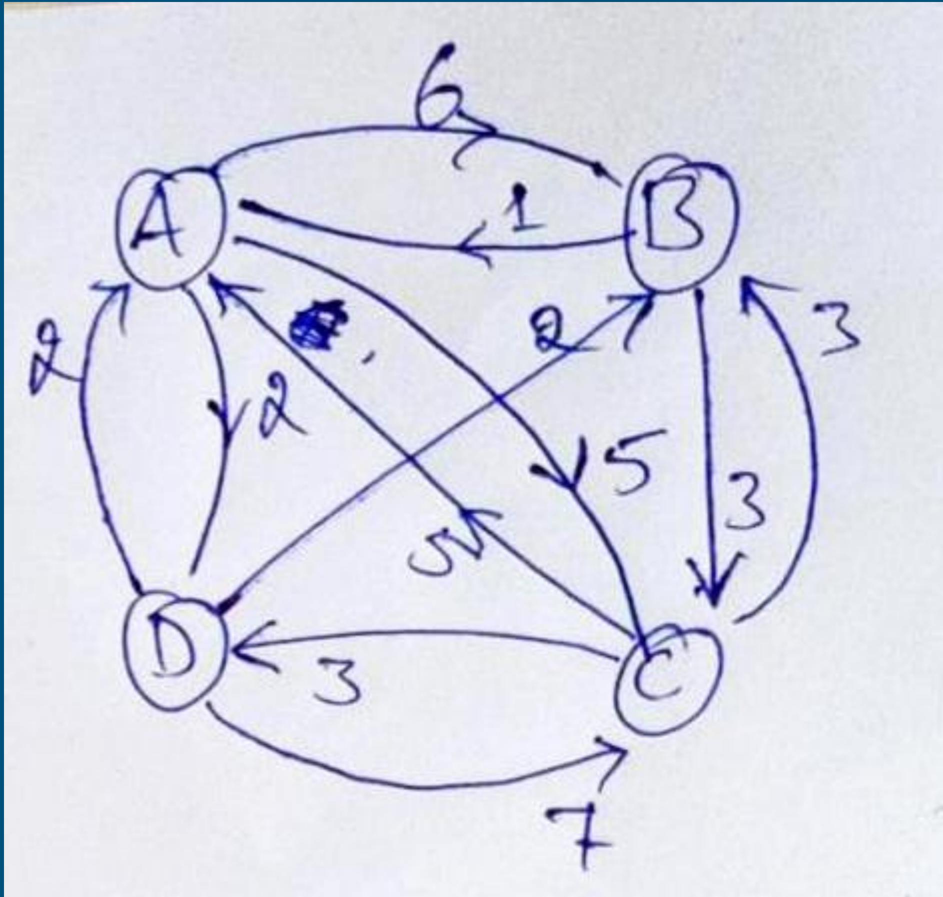
Shortest round trip: **454201** sec



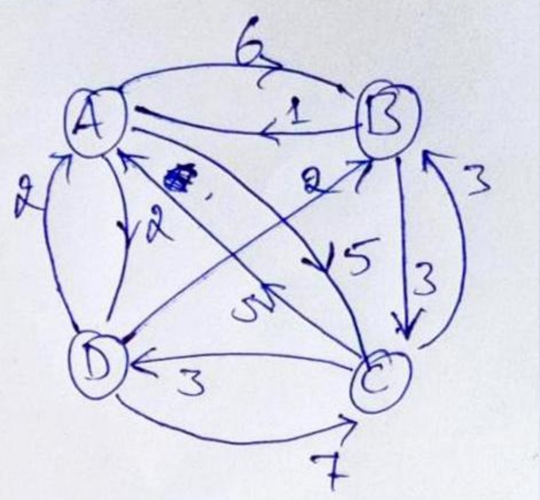
TSP - Exhaustive Search technique

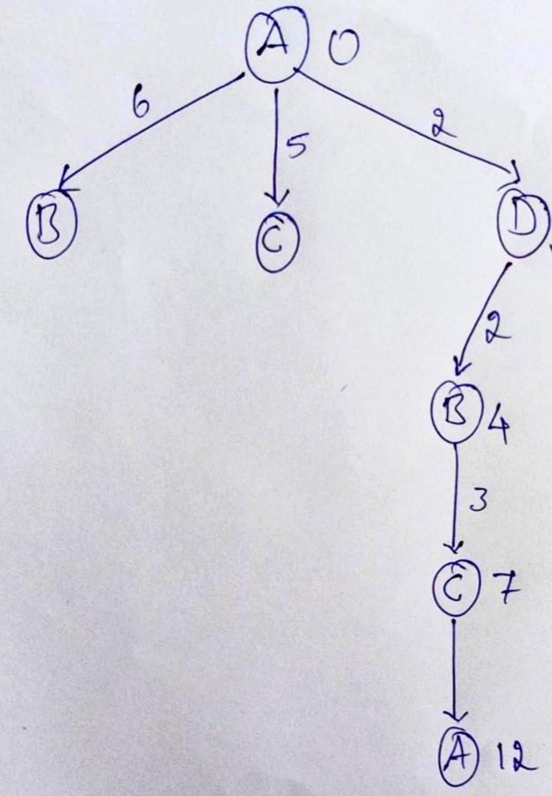
```
Algorithm TSP
mincost ← INFINITY
Permutation[1..n-1] ← [1,2,3,…,n-1] //1st permn.
do
    cost ← A[0, Permutation[1]] //1st edge of the circuit
    for i ← 1 to n-2
        cost ← cost + A[Permutation[i], Permutation[i+1]]
    cost ← cost + A[Permutation[n-1], 0] //last edge
    if (cost < mincost) mincost ← cost
while(getNextPermutation(Permutation[1..n-1]))
return mincost
```

Traveling Salesperson Problem (TSP) - Example

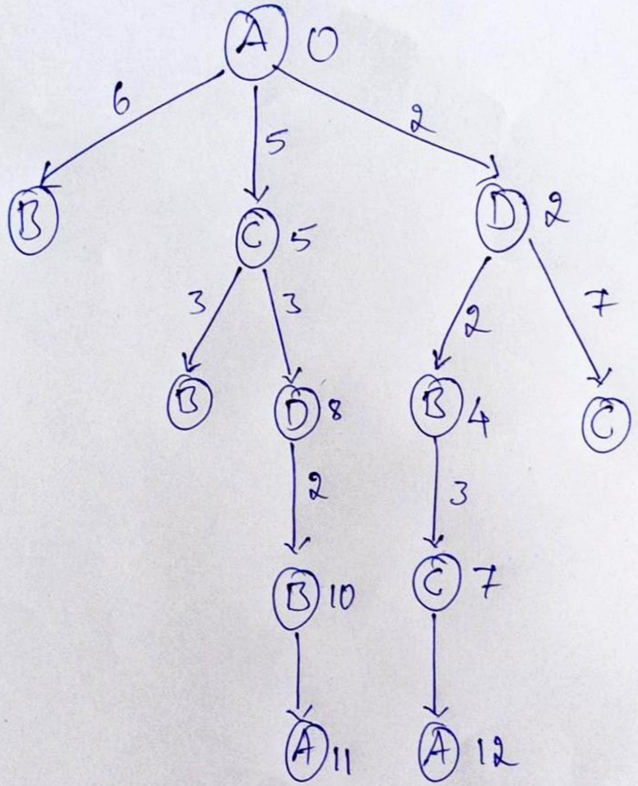


TSP - Greedy Approach

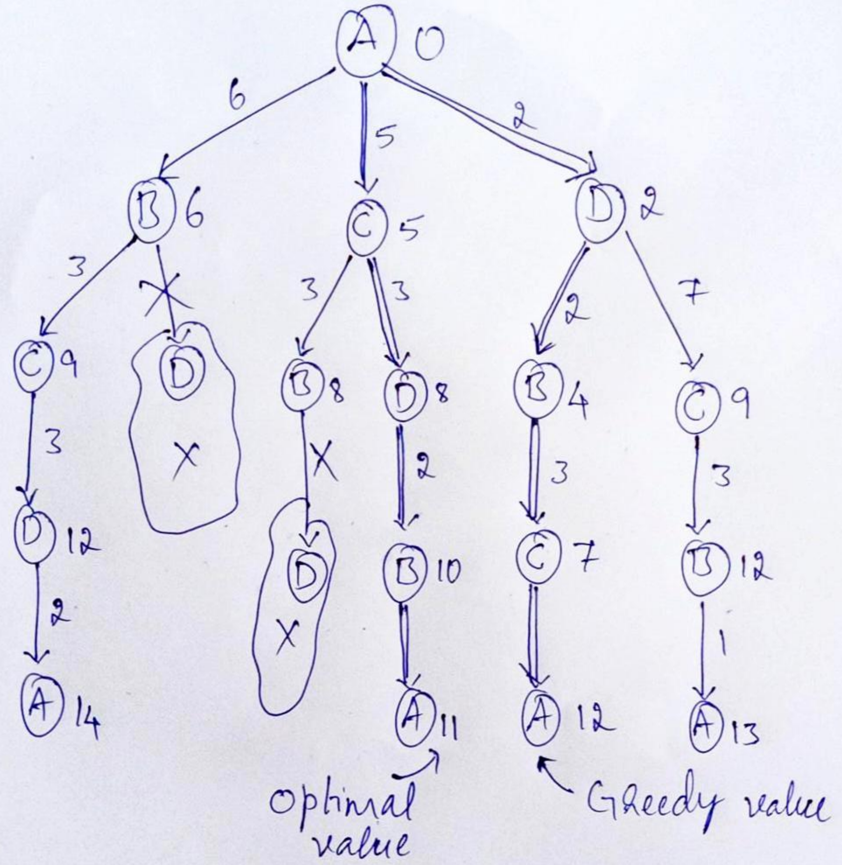




TSP - Greedy Approach



TSP - Backtracking



TSP - Branch and Bound

Assignment Problem - Backtracking & Branch-and-Bound

Objective is to **minimize** the cost of the assignment. The effort of minimizing does not help if the lower bound is already higher than a solution found so far at an intermediate stage.

$$
C = \begin{array}{c} \\ \\ \\ \\ \end{array}
\begin{array}{cccc} \text{job 1} & \text{job 2} & \text{job 3} & \text{job 4} \\ \end{array}
$$

$$
\begin{array}{cc}
C = \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix} &
\begin{array}{l} \text{person } a \\ \text{person } b \\ \text{person } c \\ \text{person } d \end{array}
\end{array}
$$