UNIVERSITY INSTITUTE OF COMPUTING

MASTER OF COMPUTER APPLICATIONS

Design and Analysis of Algorithms

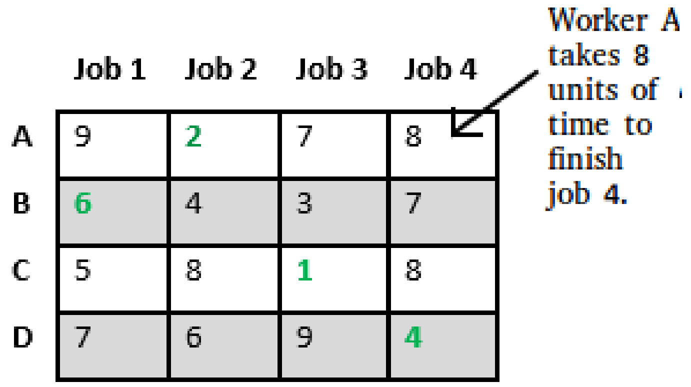
24CAP-611

UNIT-3

DISCOVER . LEARN . EMPOWER

Job Assignment Problem using Branch And Bound

- Let there be N workers and N jobs. Any worker can be assigned to perform any job, incurring some cost that may vary depending on the work-job assignment. It is required to perform all jobs by assigning exactly one worker to each job and exactly one job to each agent in such a way that the total cost of the assignment is minimized.
-



	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

Worker A takes 8 units of time to finish job 4.

An example job assignment problem. Green values show optimal job assignment that is A-Job4, B-Job1 C-Job3 and D-Job4

Job Assignment Problem using Branch And Bound

- Let us explore all approaches for this problem.
- **Solution 1: Brute Force**
  We generate n! possible job assignments and for each such assignment, we compute its total cost and return the less expensive assignment. Since the solution is a permutation of the n jobs, its complexity is O(n!).
- **Solution 2: Hungarian Algorithm**
  The optimal assignment can be found using the Hungarian algorithm. The Hungarian algorithm has worst case run-time complexity of O(n^3).

Solution 3: DFS/BFS on state space tree

A state space tree is a N-ary tree with property that any path from root to leaf node holds one of many solutions to given problem. We can perform depth-first search on state space tree and but successive moves can take us away from the goal rather than bringing closer.

The search of state space tree follows leftmost path from the root regardless of initial state. An answer node may never be found in this approach. We can also perform a Breadth-first search on state space tree. But no matter what the initial state is, the algorithm attempts the same sequence of moves like DFS.

Solution 4: Finding Optimal Solution using Branch and Bound

The selection rule for the next node in BFS and DFS is “blind”. i.e. the selection rule does not give any preference to a node that has a very good chance of getting the search to an answer node quickly. The search for an optimal solution can often be speeded by using an “intelligent” ranking function, also called an approximate cost function to avoid searching in sub-trees that do not contain an optimal solution. It is similar to BFS-like search but with one major optimization. Instead of following FIFO order, we choose a live node with least cost. We may not get optimal solution by following node with least promising cost, but it will provide very good chance of getting the search to an answer node quickly.

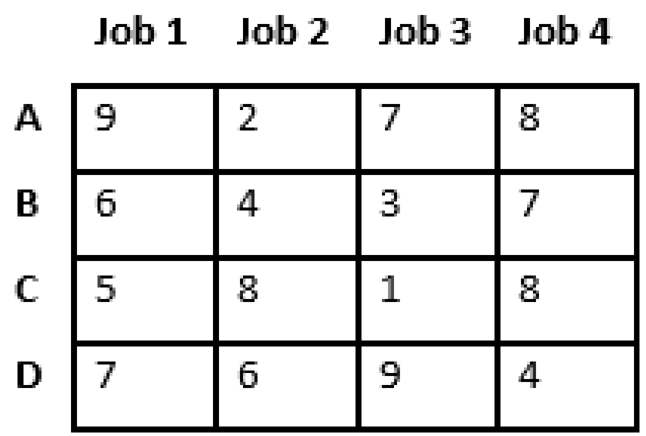
There are two approaches to calculate the cost function:

For each worker, we choose job with minimum cost from list of unassigned jobs (take minimum entry from each row).

For each job, we choose a worker with lowest cost for that job from list of unassigned workers (take minimum entry from each column).

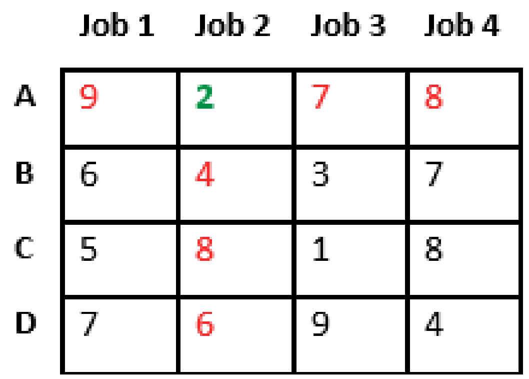
Example

Let’s take below example and try to calculate promising cost when Job 2 is assigned to worker A.



	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

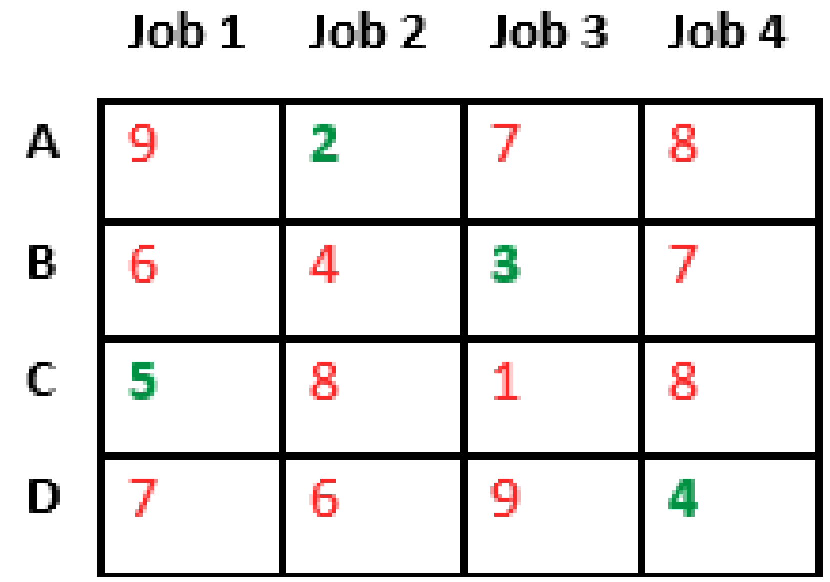
Since Job 2 is assigned to worker A (marked in green), cost becomes 2 and Job 2 and worker A becomes unavailable (marked in red).



	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

Example

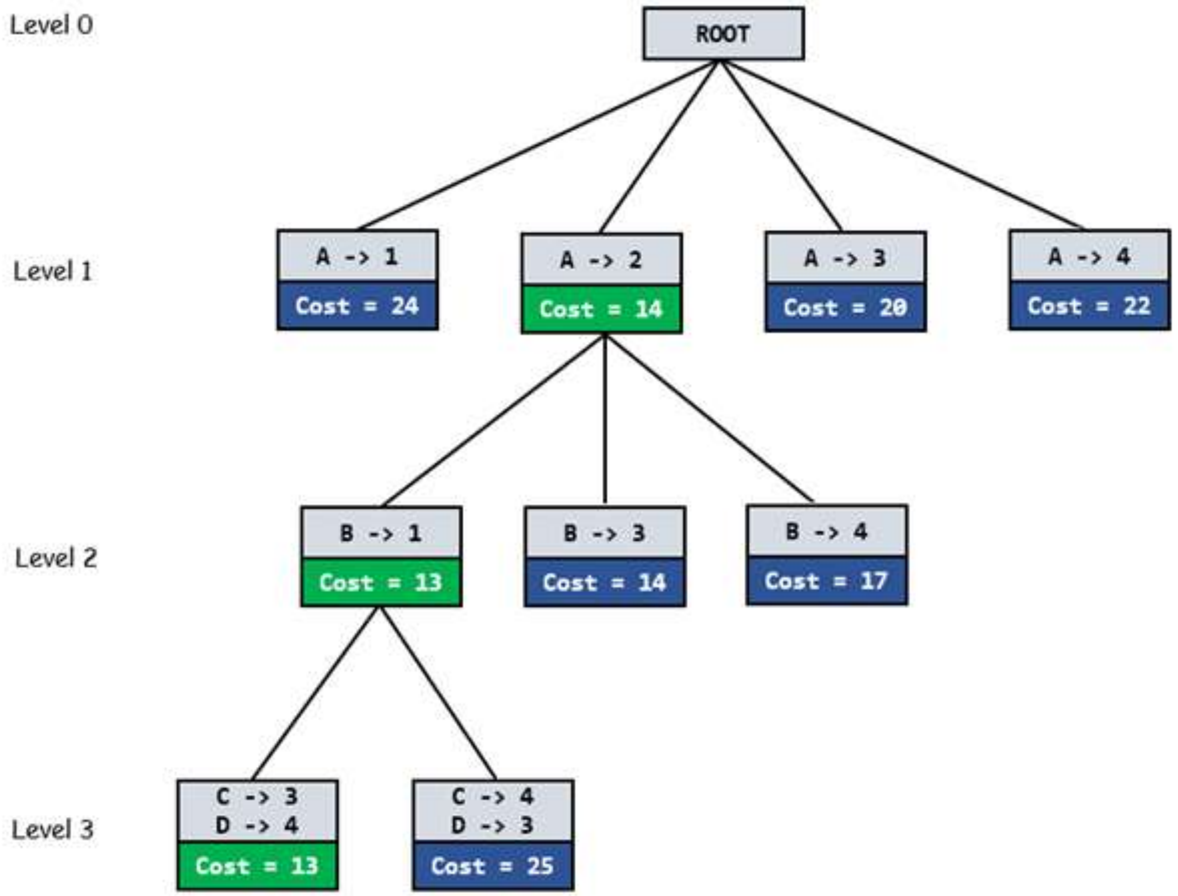
- Finally, job 1 gets assigned to worker C as it has minimum cost among unassigned jobs and job 4 gets assigned to worker C as it is only Job left. Total cost becomes 2 + 3 + 5 + 4 = 14.



	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

Problem

Below diagram shows complete search space diagram showing optimal solution path in green.



Complete Algorithm:

```
/* findMinCost uses Least() and Add() to maintain the list of
live nodes
 Least() finds a live node with least cost, deletes it from the list
and returns it
Add(x) calculates cost of x and adds it to the list of live nodes
Implements list of live nodes as a min heap */
```

Algorithm

- // Search Space Tree Node
- Node
- {
- int job_number;
- int worker_number;
- node parent;
- int cost;
- }

Algorithm

- // Input: Cost Matrix of Job Assignment problem
- // Output: Optimal cost and Assignment of Jobs
- algorithm findMinCost (costMatrix mat[][])
- {
- // Initialize list of live nodes (min-Heap)
- // with root of search tree i.e. a Dummy node
- while (true)
- {
- // Find a live node with least estimated cost
- E = Least();
- // The found node is deleted from the list
- // of live nodes
- if (E is a leaf node)
- {
- printSolution();
- return;
- }

Algorithm

- for each child x of E
- {
- Add(x); // Add x to list of live nodes;
- x->parent = E;
- // Pointer for path to root
- }
- }
- }

References

1) https://www.tutorialspoint.com/data_structures_algorithms/divide_and_conquer.htm

2) Data Structures and Algorithms made easy By Narasimha Karumanchi.

3) The Algorithm Design Manual, 2nd Edition by Steven S Skiena

4) Fundamentals of Computer Algorithms - Horowitz and Sahani

CU
CHANDIGARH UNIVERSITY

NAAC GRADE A+

THANK YOU