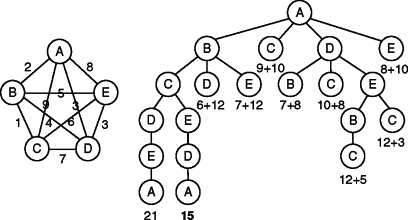
**Introduction :** Branch-and-bound  is a general technique for improving the searching process by systematically enumerating all candidate solutions and disposing of obviously impossible solutions.

Branch-and-bound usually applies to those problems that have finite solutions, in which the solutions can be represented as a sequence of options. The first part of branch-and-bound, **branching**, requires several choices to be made so that the choices branch out into the solution space. In these methods, the solution space is organized as a treelike structure.

**Problem Statement :**

*Branch-and-bound* (BnB) is a general programming paradigm used, for example, in operations research to solve hard combinatorial optimization problems. *Branching* is the process of spawning sub-problems, and *bounding* refers to ignoring partial solutions that cannot be better than the current best solution. To this end, lower and upper bounds *L* and *U* are maintained. Since global control values on the solution quality improve over time, branch-and-bound is effective in solving *optimization problems*, in which a cost-optimal assignment to the problem variables has to be found.

* Figure shows an instance of TSP ([Travelling Salesman Problem using Branch And Bound](https://tutorialspoint.dev/algorithm/branch-and-bound-algorithm/branch-bound-set-5-traveling-salesman-problem)) and a solution tree, which is constructed by making choices on the next cities to visit.

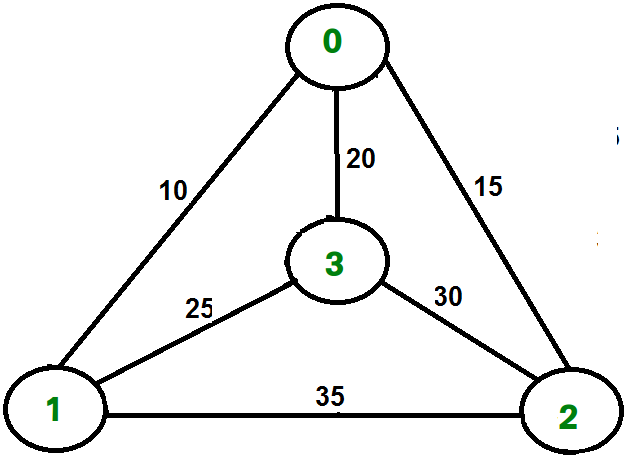
Methods used to Solve Branch and Bound

* [0/1 Knapsack using Branch and Bound](https://tutorialspoint.dev/algorithm/branch-and-bound-algorithm/branch-and-bound-set-1-introduction-with-01-knapsack)
* [Implementation of 0/1 Knapsack using Branch and Bound](https://tutorialspoint.dev/algorithm/branch-and-bound-algorithm/branch-and-bound-set-2-implementation-of-01-knapsack)
* [8 puzzle Problem using Branch And Bound](https://tutorialspoint.dev/algorithm/branch-and-bound-algorithm/branch-bound-set-3-8-puzzle-problem)
* [Job Assignment Problem using Branch And Bound](https://tutorialspoint.dev/algorithm/branch-and-bound-algorithm/branch-bound-set-4-job-assignment-problem)
* [N Queen Problem using Branch And Bound](https://tutorialspoint.dev/algorithm/branch-and-bound-algorithm/branch-and-bound-set-4-n-queen-problem)
* [Travelling Salesman Problem using Branch And Bound](https://tutorialspoint.dev/algorithm/branch-and-bound-algorithm/branch-bound-set-5-traveling-salesman-problem)

**Branch and cut** method is a very successful **algorithm** for solving a variety of integer programming problems, and it also can provide a guarantee of optimality. This method are exact **algorithm** consisting of a combination of a **cutting** plane method and a **branch**-and-bound **algorithm.**

**Algorithm : Travelling Salesman Problem using Branch And Bound**

Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible tour that visits every city exactly once and returns to the starting point.



For example, consider the graph shown in figure on right side. A TSP tour in the graph is 0-1-3-2-0. The cost of the tour is 10+25+30+15 which is 80.

We have discussed following solutions  
1) [Naive and Dynamic Programming](https://tutorialspoint.dev/slugresolver/travelling-salesman-problem-set-1/)  
2) [Approximate solution using MST](https://tutorialspoint.dev/slugresolver/travelling-salesman-problem-set-2-approximate-using-mst/)

**Branch and Bound Solution**  
As seen in the previous articles, in Branch and Bound method, for current node in tree, we compute a bound on best possible solution that we can get if we down this node. If the bound on best possible solution itself is worse than current best (best computed so far), then we ignore the sub-tree rooted with the node.

Note that the cost through a node includes two costs.  
1) Cost of reaching the node from the root (When we reach a node, we have this cost computed)  
2) Cost of reaching an answer from current node to a leaf (We compute a bound on this cost to decide whether to ignore sub-tree with this node or not).

* In cases of a **maximization problem**, an upper bound tells us the maximum possible solution if we follow the given node. For example in [0/1 knapsack we used Greedy approach to find an upper bound](https://tutorialspoint.dev/slugresolver/branch-and-bound-set-2-implementation-of-01-knapsack/).
* In cases of a **minimization problem**, a lower bound tells us the minimum possible solution if we follow the given node. For example, in [Job Assignment Problem](https://tutorialspoint.dev/slugresolver/branch-bound-set-4-job-assignment-problem/), we get a lower bound by assigning least cost job to a worker.

In branch and bound, the challenging part is figuring out a way to compute a bound on best possible solution. Below is an idea used to compute bounds for Travelling salesman problem.

*Cost of any tour can be written as below.*

*Cost of a tour T = (1/2) \* &Sum; (Sum of cost of two edges*

*adjacent to u and in the*

*tour T)*

*where u ∈ V*

*For every vertex u, if we consider two edges through it in T,*

*and sum their costs. The overall sum for all vertices would*

*be twice of cost of tour T (We have considered every edge twice.)*

*(Sum of two tour edges adjacent to u) >= (sum of minimum weigh two edges adjacent to u)*

*Cost of any tour >= 1/2) \* &Sum; (Sum of cost of two minimum weight edges adjacent to u)*

*where u ∈ V*

For example, consider the above shown graph. Below are minimum cost two edges adjacent to every node.

*3 (0, 3), (1, 3) 45*

*Thus a lower bound on the cost of any tour = 1/2(25 + 35 + 45 + 45)*

*= 753 (0, 3), (1, 3) 45*

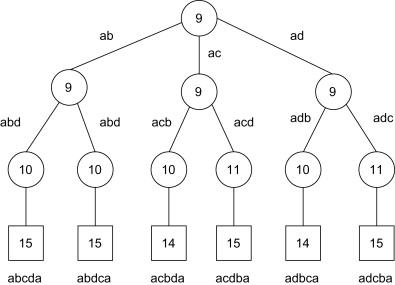
*Thus a lower bound on the cost of any tour = 1/2(25 + 35 + 45 + 45)*

*= 75*

**Branch and bound** is an algorithm design paradigm which is generally used for solving combinatorial optimization problems. These problems are typically exponential in terms of time complexity and may require exploring all possible permutations in worst case. The Branch and Bound Algorithm technique solves these problems relatively quickly.

For applying branch-and-bound search to general state space problems, we concentrate on DFS extended with upper and lower bounds. In this context, branching corresponds to the generation of successors, so that DFS can be casted as generating a *branch-and-bound search tree*. We have already seen that one way of obtaining a lower bound *L* for the problem state *u* is to apply an admissible heuristic *h*, or L(u)=g(u)+h(u) for short. An initial upper bound can be obtained by constructing any solution, such as one established by a greedy approach.

As with standard DFS, the first solution obtained might not be optimal. With DFBnB, however, the solution quality improves over time together with the global value *U* until eventually the lower bound L(u) at some node *u* is equal to *U*. In this case an optimal solution has been found, and the search terminates.

The implementation of DFBnB is shown in Algorithm. At the beginning of the search, the procedure is invoked with the start node and with the upper bound *U* set to some reasonable estimate (it could have been obtained using some heuristics; the lower it is, the more can be pruned from the search tree, but in case no upper bound is known, it is safe to set it to ∞). A global variable *best-Path* keeps track of the actual solution path. The recursive search routine is depicted in Algorithm .

Sorting the set of successors according to increasing *L*-values is an optional refinement to the algorithm that often aids in accelerating the search for finding an early solution.