Categorizing Algorithms

Mohammad Tehranipoor
ECE Department

Greedy Algorithms

- **Prim’s Algorithm** (Minimum Spanning Tree)
  - A subgraph that is a tree and that spans (reaches out to) all vertices of the original graph is called a spanning tree.

- **Kruskal’s Algorithm** (Minimum Spanning Tree)

- **Dijkstra’s Algorithm** (Shortest Path)
  - Find the best way of getting from \( s \) to \( t \) where \( s \) and \( t \) are vertices in a graph.
Divide and Conquer Algorithms

- **Merge Sort**
  - A sort algorithm that splits the items to be sorted into two groups, recursively sorts each group, and merges them into a final, sorted sequence.

- **Quick Sort**
  - Pick an element from the array (the pivot), partition the remaining elements into those greater than and less than this pivot, and recursively sort the partitions.

- **Closest Pair**
  - Find a pair of points with the smallest distance between them.

Branch and Bound Algorithms

- **Traveling Salesman Problem (TSP)**
  - Given a finite number of "cities" along with the cost of travel between each pair of them, find the cheapest way of visiting all the cities and returning to your starting point.

- **Integer Linear Programming (ILP)**

Graphs

- Prim’s Algorithm
- Kruskal’s Algorithm
- ...
Cont.

- **State Search Algorithms**
  - Travelling Salesman Problem (TSP)
  - Integer Linear Programming (ILP)

- **Dynamic Programming Algorithms**
  - Fibonacci Numbers
  - Shortest Paths

- **Backtracking Algorithms**
  - TSP

---

**CAD Algorithms**

**Branch and Bound Algorithms**

Mohammad Tehranipoor  
ECE Department
Branch and Bound

- **Definition:**
  - An algorithm technique to find the optimal solution by *keeping the best solution found so far*. If a partial solution cannot improve on the best, it is abandoned.

- Branch and Bound is a general search method.

- Branch and Bound is a tree of trees based idea, in which the tree of all possible trees is constructed with the actual trees with \( n \) leaves of the tree, and we will be looking at subtrees in order to prune off unnecessary parts of the tree-space.

---

Example

- Suppose we want to find the shortest path from **A** to **B**, and the shortest path found until now is 400 miles.

- Suppose we are to next consider path through **C**. If the shortest distance from **A** to **C** is 400 miles, there is no reason to explore possible roads from **C**; they will be at least 410 miles (350+60), which is worse than the shortest known path. So, we need not explore paths from **C**.

**Shortest Path Problem**
Branch and Bound Algorithms

- Branch and Bound Algorithms:
  - **Travelling Salesman Problem**
    - given a finite number of "cities" along with the cost of travel between each pair of them, find the cheapest way of visiting all the cities and returning to your starting point.
  - **Job Scheduling**
  - **Integer Linear Programming (ILP)**

---

Example

- Consider the following problem from machine scheduling.
- We have \( n \) jobs each of which can be processed either on machine 1 or machine 2.
- But the processing times may differ depending on the machine and the job is to be processed.
- Suppose we have four jobs with the following processing times:

<table>
<thead>
<tr>
<th></th>
<th>Job 1</th>
<th>Job 2</th>
<th>Job 3</th>
<th>Job 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine 1</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Machine 2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

**Minimize the completion time of the job that is processed last**

Integer Linear Programming can solve this problem.
In the total enumeration tree, at any node, we can show that the optimal solution cannot occur in any of its descendents, then there is no need for us to consider those descendendent nodes.

We can prune the tree at the node.

This may reduce the problem to a computationally manageable size.

The branch and bound approach is not a heuristic, or approximating procedure.

It is an exact optimizing procedure that finds the optimal solution.
Divide-and-Conquer

Divide-and-conquer is a top-down technique for designing algorithms that consists of dividing the problem into smaller subproblems hoping that the solutions of the subproblems are easier to find and then composing the partial solutions into the solution of the original problem.
Definition

Definition:

An algorithmic technique to solve a problem on an instance of size $n$, a solution is found either directly because solving that instance is easy (typically, because the instance is small) or the instance is divided into two or more smaller instances. Each of these smaller instances is recursively solved, and the solutions are combined to produce a solution for the original instance.

Divide-and-Conquer

Divide-and-conquer paradigm consists of following major phases:

- Breaking the problem into several sub-problems that are similar to the original problem but smaller in size.
- Solve the sub-problem recursively.
- Combine these solutions to subproblems to create a solution to the original problem.
Examples

- This technique yields elegant, simple and quite often very efficient algorithms.

- **Merge sort**
  - A sort algorithm that splits the items to be sorted into two groups, recursively sorts each group, and merges them into a final, sorted sequence.

Examples

- **Quick sort**
  - Pick an element from the array (the pivot), partition the remaining elements into those greater than and less than this pivot, and recursively sort the partitions.
Example 3

- **Closest Pair**
  - Given a set of \( n \) points \((x_i, y_i)\), the problem is to find distance between the two closest points.

- **Binary Search**
  - Simplest application of divide-and-conquer
  - Given a name and number of names in phonebook, the problem is to find the phone number associated with the name.

<table>
<thead>
<tr>
<th>Name</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name 1</td>
<td>Number 1</td>
</tr>
<tr>
<td>Name 2</td>
<td>Number 2</td>
</tr>
<tr>
<td>Name n</td>
<td>Number n</td>
</tr>
</tbody>
</table>