Greedy Algorithms

This document contains summary of some of the items discussed as part of the COMS 3110 Lectures. This document **does not replace the lecture materials**. This document may contain some topics that are not covered as part of the lecture; you will not be tested on those parts, they are made available to you for gaining further knowledge on topics/concepts that are related to class-lecture.

1 Topics Covered

- Greedy algorithms
- Single Source Shortest Path Algorithm due to Dijkstra

2 Greedy Algorithms

Greedy algorithms are a class of algorithms that make locally optimal choices at each step with the hope of finding a global optimum. The correctness of greedy algorithms is often proven using arguments such as the **Greedy Stays Ahead Argument** and the **Exchange Argument**.

- **Greedy Stays Ahead Argument:**
  This argument is based on the idea that at any point in the algorithm, the greedy choice "stays ahead" of the optimal solution. This means that the greedy solution is always as good as or better than the optimal solution at each step. In other words, the greedy choice is "ahead" of the optimal solution at every point in time, and this ensures that the greedy approach eventually leads to an optimal solution.
- **Exchange Argument:**
  The exchange argument is a proof technique used to show the correctness of greedy algorithms. It involves comparing the greedy solution to an optimal solution and showing that by swapping (or exchanging) certain elements of the optimal solution with those of the greedy solution, you can produce a solution that is still optimal. This process shows that the greedy solution is at least as good as the optimal solution. The idea is to "exchange" parts of the optimal solution for parts of the greedy solution, ensuring that the result remains optimal.

3 Single Source Shortest Path Algorithm due to Dijkstra

Basic Definition & Background. In this section, we will consider weighted graphs, where each edge in the graph is associated with a weight, whose valuation is determined by a weight function. For any edge $e = (u, v)$, its weight is denoted by $wt(e)$. Recall that, a path graph is a sequence of vertices, where each successive vertices are connected by an edge relation. In the context of the weighted graph, we will say that a weight or length of a path is the sum of the weights of the edges that form the path. For instance, let $\pi = v_1, v_2, v_3, \ldots, v_k$ be a path from vertex v_1 to vertex v_k; then the length of the path is

$$\sum_{i=1}^{k-1} wt(v_i, v_{i+1})$$

One of the objectives is to find the shortest length path to all vertices from a specified vertex. This is referred to as the single source shortest path problem. Note that, when the weights of all edges are identical, we can apply BFS exploration strategy to address this problem. However, in the context of weights, pure BFS strategy is not a correct strategy. For instance, consider a graph, where v_0 has an edge v_1 and v_2, with edge weights 10 and 5, respectively; and v_2 has an edge to v_1 with edge weight 2. In this case, the shortest path

distance from v_0 to v_1 is via v_2, with the distance being $5 + 2 = 7$ (which is strictly less than the directed edge from v_0 to v_1, a solution that would have been obtained using pure BFS).

We will first consider this problem in the context where all weights of **non-negative**. The solution to this problem is due to E. Dijkstra (Dijkstra algorithm). This algorithm uses two properties that hold for the problem:

1. Optimal substructure property: For any shortest path from u to v, if there is an intermediate vertex w, then subpath u to w and subpath w to v are shortest paths.

2. Greedy choice property: Consider S is a set of vertices for which the shortest distances $\delta(v)$ from source vertex is known. For all vertices $u \in V/S$ let $d(u) = min_{v \in S}\{\delta(v) + wt(u, v)\}$. Then, the vertex x with the minimal d-value is the one where $d(x) = \delta(x)$. Intuitively, one can greedily choose the the vertex that is closest to the source and conclude the shortest distance of the vertex. The proof of this claim depends on the negative weight values. The proof has been done in class.

Dijkstra algorithm with min-heap implementation of priority queue

```
Input G = (V, E, wt) and source s.

1. Initialize d(v) for all vertices to infinity, and v.parent = undefined;
2. d(s) = 0; s.parent = 0;
3. add all vertices to a minheap H with key being d-value
4. Initialize v.inQ = true for all vertices.
5. while H != empty
6.    v = extractMin(H);
7.    v.inQ = false;
8.    for all (v, v') in E and v'.inQ == true
9.        if (d(v') > d(v) + wt(v, v') then
10.            d(v') = d(v) + wt(v, v')
11.           v'.parent = v;
12.           updateHeap(H, v', d(v'))
```

The objective of function `updateHeap(H, v', d(v'))` is to bring v' to its correct position (using the standard heapifyUp subroutine) in the heap as the $d(v')$ valuation decreases in the line 10. This requires that we keep track of the indices of each vertex in the array implementation of the heap. One can realize that by setting a map-structure, M such that $M(v)$ returns the index of v in the array implementation of heap.

The parent-property associated with the vertices can be used to generate a shortest path from source to any vertex in the graph and the d-property captures the length of such shortest path.

The runtime of the algorithm $O((V + E)logV)$, where the factor $log(V)$ is due to the heap operations: extractMin and updateHeap.