CS 225 Spring 2019 :: TA Lecture Notes
4/22 Prim's Algorithm & Dijkstra

By Wenjie

- **Prim's Algorithm**

```
1  PrimMST(G, s):
2    Input: G, Graph;
3           s, vertex in G, starting vertex
4    Output: T, a minimum spanning tree (MST) of G
5
6    foreach (Vertex v : G):
7      d[v] = +inf
8      p[v] = NULL
9    d[s] = 0
10
11   PriorityQueue Q   // min distance, defined by d[v]
12   Q.buildHeap(G.vertices())
13   Graph T           // "labeled set"
14
15   repeat n times:
16     Vertex m = Q.removeMin()
17     T.add(m)
18     foreach (Vertex v : neighbors of m not in T):
19       if cost(v, m) < d[v]:
20         d[v] = cost(v, m)
21         p[v] = m
22
23   return T
```

- Runtime Analysis
  - Lines 6 to 10 → it is a regular for loop that goes through vertice; it takes O(n) time.
  - Lines 12 to 15 → again O(n) time.
  - Lines 15 to 22 → this is the body of another for loop that loops in O(n) time. Inside of the for loop we have:
    - Line 16 → In case we are using a heap, remove takes log(n) time because of heapify down.
    - Line 17 → In case of adjacency matrix implementation, add vertex takes O*(n) time.
    - The dominant term here is O*(n) and since it is inside of a for loop, the total time will be $n \times O^*(n) = O(n^2)$.

- Line 19 → O(m).
- Lines 20 to 22 → we need to update the heap which takes log(n) time.
- The time to execute 19 to 22 within a loop is $O(m \cdot log(n))$.

- Overall the running time for Prim's algorithm when using a heap and adjacency matrix is $O(n^2 + m \cdot log(n))$.

	Adjacency Matrix	Adjacency List
Heap	$O(n^2 + m \cdot log(n))$ $m \sim [n, n^2]$	
Unsorted Array		

- The reason we have n^2 in the running time is because adding a vertex takes O(n). Therefore, we consider adjacency list → 16-17 will run in $n \times log(n)$.

	Adjacency Matrix	Adjacency List
Heap	$O(n^2 + m \cdot log(n))$	$O(n \cdot log(n) + m \cdot log(n))$
Unsorted Array		

- Notice that we update the heap quite a bit. How can we reduce the cost of updating? - We can use an unsorted array.
  - Line 16 → remove takes O(n) because we need to loop over the whole array to find the vertex to remove.
  - Line 17 → O(n) as previously explained.
  - Lines 19 to 22 → will now take O(m) because we don't need to update anything, we are just looping over edges.
  - Total running time will be $O(n^2)$.

	Adjacency Matrix	Adjacency List
Heap	$O(n^2 + m \cdot log(n))$	$O(n \cdot log(n) + m \cdot log(n))$

CS 225 Spring 2019 :: TA Lecture Notes
4/22 Prim's Algorithm & Dijkstra

By Wenjie

Unsorted Array	$O(n^2 + m) = O(n^2)$	$O(n^2)$

- Based on the analysis, we can see that we should never use heap with the adjacency matrix. Everything else seems reasonable depending on the data.
  - Case 1: the data is sparse → use (heap + adj list) and the running time will be $O(n \cdot log(n))$.
  - Case 2: the data is dense → use (unsorted array + adj matrix/list) and the running time will be $O(n^2)$.

- MST Algorithms Running Times

Kruskal's Algorithm	Prim's Algorithm
$O(n + m \cdot log(n))$	$O(n \cdot log(n) + m \cdot log(n))$

- With MSTs, we are assuming that the graph is connected and that it has at least $n-1$ edges, $m \geq n-1$.
- Therefore, O(n) is O(m), but not the opposite.

Kruskal's Algorithm	Prim's Algorithm
$O(m \cdot log(n))$	$O(m \cdot log(n))$

- There is a way to make Prim's algorithm faster. We can use something called Fibonacci Heap to get $O(n \cdot log(n) + m)$ time, but we will not cover that in this class.