Data Structures and Algorithms

Lecture 28

Aniket Basu Roy

2026-04-06 Mon

Contents

1 Agenda

1.1 Time Complexity of Prim-Jarník Algorithm

- Recap of algorithm and key-array invariant
- Min-heap implementation details

- Operation counts and costs
- Total time complexity
- Comparison: array vs. heap vs. Fibonacci heap

2 Recap: Prim-Jarník Algorithm

```
PRIM(G, w, r)
1.  for each u in G.V
2.      u.key = infinity
3.      u.parent  = NIL
4.  r.key = 0
5.  Q = G.V               // min-priority queue keyed by u.key
6.  while Q != {}
7.      u = EXTRACT-MIN(Q)
8.      for each v in G.Adj[u]
9.          if v in Q and w(u,v) < v.key
10.             v.parent   = u
11.             v.key = w(u,v)   // DECREASE-KEY in Q
```

Let $n = |V|$, $m = |E|$ (connected graph, so $m \geq n - 1$).

3 Key-Array Invariant

For each $v \notin S$ (not yet extracted):

$$v.\text{key} = \min \{\, w(u, v) \mid u \in S,\, (u, v) \in E \,\}$$

or ∞ if no such edge exists.

This value changes as S grows: when a new vertex u is added to S, we update v.key for all $v \in \text{Adj}(u) \cap Q$.

4 Min-Heap Implementation

Implement Q as a **min-heap** keyed by v.key.

4.1 Auxiliary structures

- Boolean array inQ[v]: TRUE iff $v \in Q$. Initialized to TRUE; set FALSE on EXTRACT-MIN.
- Position map pos[v]: index of v in the heap array. Needed to locate v in $O(1)$ for DECREASE-KEY.

4.2 Operations used

- BUILD-MIN-HEAP: initialize heap from all n vertices.
- EXTRACT-MIN: remove and return the vertex with smallest key.
- MIN-HEAP-DECREASE-KEY(Q, v, k): update v.key $\leftarrow k$ and restore heap property (bubble up).

5 Operation Counts and Costs

Operation	# Calls	Cost per call	Total
BUILD-MIN-HEAP	1	$O(n)$	$O(n)$
EXTRACT-MIN	n	$O(\log n)$	$O(n \log n)$
MIN-HEAP-DECREASE-KEY	$\leq m$	$O(\log n)$	$O(m \log n)$

5.1 Why $\leq m$ DECREASE-KEY calls?

Each edge (u, v) can trigger a DECREASE-KEY on v only when u is extracted. Each edge is processed once per endpoint extraction, so at most m updates total.

5.2 Total

$$O(n) + O(n \log n) + O(m \log n) \;=\; O((n+m) \log n) \;=\; O(m \log n).$$

(Since $m \geq n - 1$, the $m \log n$ term dominates.)

$$\boxed{T_{\text{Prim-Jarník (min-heap)}} = O(m \log n)}$$

6 Array-Based Priority Queue

Implement Q as an unsorted array of size n:

- EXTRACT-MIN: scan all entries in Q to find the minimum — $O(n)$ per call.
- DECREASE-KEY: update a single entry — $O(1)$ per call.

Step	Cost
n EXTRACT-MIN calls	$O(n^2)$
$\leq m$ DECREASE-KEY	$O(m)$
Total	$O(n^2 + m)$

For dense graphs ($m = \Theta(n^2)$): $O(n^2)$, better than the heap-based $O(m \log n) = O(n^2 \log n)$.

7 Summary: Implementation Comparison

Priority Queue	EXTRACT-MIN	DECREASE-KEY	Total
Array (unsorted)	$O(n)$	$O(1)$	$O(n^2)$
Binary min-heap	$O(\log n)$	$O(\log n)$	$O(m \log n)$

- Sparse graphs ($m = O(n)$): binary min-heap gives $O(n \log n)$.
- Dense graphs ($m = \Theta(n^2)$): array gives $O(n^2)$ (optimal for dense case).

8 Questions

- For a graph with $m = O(n^{1.5})$, which implementation is better: array or binary heap?