15.082J & 6.855J & ESD.78J

Shortest Paths 2:
Bucket implementations of Dijkstra’s Algorithm
R-Heaps
A Simple Bucket-based Scheme

Let \( C = 1 + \max(c_{ij} : (i,j) \in A) \); then \( nC \) is an upper bound on the minimum length path from 1 to \( n \).

**RECALL:** When we select nodes for Dijkstra's Algorithm we select them in increasing order of distance from node 1.

**SIMPLE STORAGE RULE.** Create buckets from 0 to \( nC \).

Let \( BUCKET(k) = \{i \in T : d(i) = k\} \). Buckets are sets of nodes stored as doubly linked lists. \( O(1) \) time for insertion and deletion.
Dial’s Algorithm

- Whenever $d(j)$ is updated, update the buckets so that the simple bucket scheme remains true.

- The FindMin operation looks for the minimum non-empty bucket.

- To find the minimum non-empty bucket, start where you last left off, and iteratively scan buckets with higher numbers.
Running time for Dial’s Algorithm

- $C = 1 + \max(c_{ij} : (i,j) \in A)$.
- Number of buckets needed: $O(nC)$
- Time to create buckets: $O(nC)$
- Time to update $d(\ )$ and buckets: $O(m)$
- Time to find min: $O(nC)$
- Total running time: $O(m + nC)$.

This can be improved in practice; e.g., the space requirements can be reduced to $O(C)$. 
Additional comments on Dial’s Algorithm

- Create buckets when needed. Stop creating buckets when each node has been stored in a bucket.

- Let $d^* = \max \{d^*(j) : j \in N\}$. Then the maximum bucket ever used is at most $d^* + C$.

Suppose $j \in \text{Bucket}(d^* + C + 1)$ after update$(i)$. But then $d(j) = d(i) + c_{ij} \leq d^* + C$
A 2-level bucket scheme

- Have two levels of buckets.
  - Lower buckets are labeled 0 to K-1 (e.g., K = 10)
  - Upper buckets all have a range of K. First upper bucket’s range is K to 2K – 1.
  - Store node j in the bucket whose range contains d(j).
Find Min

- FindMin consists of two subroutines
  - SearchLower: This procedure searches lower buckets from left to right as in Dial’s algorithm. When it finds a non-empty bucket, it selects any node in the bucket.
  
  - SearchUpper: This procedure searches upper buckets from left to right. When it finds a bucket that is non-empty, it transfers its elements to lower buckets.

FindMin

If the lower buckets are non-empty, then SearchLower; Else, SearchUpper and then SearchLower.
More on SearchUpper

- SearchUpper is carried out when the lower buckets are all empty.

- When SearchUpper finds a non-empty bucket, it transfers its contents to lower buckets. First it relabels the lower buckets.

![Diagram of 2-Level Bucket Algorithm](image-url)
Running Time Analysis

- **Time for SearchUpper:** $O(nC/K)$
  - $O(1)$ time per bucket

- **Number of times that the Lower Buckets are filled from the upper buckets:** at most $n$.

- **Total time for FindMin in SearchLower**
  - $O(nK)$; $O(1)$ per bucket scanned.

- **Total Time for scanning arcs and placing nodes in the correct buckets:** $O(m)$

- **Total Run Time:** $O(nC/K + nK + m)$
  - Optimized when $K = C^{0.5}$
  - $O(nC^{0.5} + m)$
More on multiple bucket levels

- Running time can be improved with three or more levels of buckets.

- Runs great in practice with two levels

- Can be improved further with buckets of range (width) 1, 1, 2, 4, 8, 16 ...
  - Radix Heap Implementation
A Special Purpose Data Structure

- **RADIX HEAP**: a specialized implementation of priority queues for the shortest path problem.

- **A USEFUL PROPERTY** (of Dijkstra's algorithm): The minimum temporary label $d(\cdot)$ is monotonically non-decreasing. The algorithm labels node in order of increasing distance from the origin.

- $C = 1 + \text{max length of an arc}$
Radix Heap Example

Buckets:
- bucket sizes grow exponentially
- ranges change dynamically

Radix Heap Animation
Analysis: FindMin

- Scan from left to right until there is a non-empty bucket. If the bucket has width 1 or a single element, then select an element of the bucket.

- Time per find min: \( O(K) \), where \( K \) is the number of buckets.
Analysis: Redistribute Range

- Redistribute Range: suppose that the minimum non-empty bucket is Bucket $j$. Determine the minimum distance label $d^*$ in the bucket. Then distribute the range of Bucket $j$ into the previous $j-1$ buckets, starting with value $d^*$.
  - Time per redistribute range: $O(K)$. It takes $O(1)$ steps per bucket.
  - Time for determining $d^*$: see next slide.

$d(5) = 9$ (min label)
Analysis: Find min $d(j)$ for $j$ in bucket

- Let $b$ be the number of items in the minimum bucket. The time to find the min distance label of a node in the bucket is $O(b)$.
  - Every item in the bucket will move to a lower index bucket after the ranges are redistributed.
  - Thus, the time to find $d^*$ is dominated by the time to update contents of buckets.
  - We analyze that next
Analysis: Update Contents of Buckets

When a node j needs to move buckets, it will always shift left. Determine the correct bucket by inspecting buckets one at a time.

- O(1) whenever we need to scan the bucket to the left.
- For node j, updating takes O(K) steps in total.

\[d(5) = 9\]
Running time analysis

- FindMin and Redistribute ranges
  - $O(K)$ per iteration. $O(nK)$ in total

- Find minimum $d(j)$ in bucket
  - Dominated by time to update nodes in buckets

- Scanning arcs in Update
  - $O(1)$ per arc. $O(m)$ in total.

- Updating nodes in Buckets
  - $O(K)$ per node. $O(nK)$ in total

- Running time: $O(m + nK)$
  - $O(m + n \log nC)$

- Can be improved to $O(m + n \log C)$
Summary

- Simple bucket schemes: Dial’s Algorithm
- Double bucket schemes: Denardo and Fox’s Algorithm
- Radix Heap: A bucket based method for shortest path
  - buckets may be redistributed
  - simple implementation leads to a very good running time
  - unusual, global analysis of running time