BucketSort (aka BinSort)
If all values to be sorted are known to be between 1 and $K$, create an array \( \text{count} \) of size $K$, increment counts while traversing the input, and finally output the result.

Example $K=5$. Input = (5,1,3,4,2,1,5,4,5)

Running time to sort $n$ items?

BucketSort Complexity: $O(n+K)$
- Case 1: $K$ is a constant
  - BinSort is linear time
- Case 2: $K$ is variable
  - Not simply linear time
- Case 3: $K$ is constant but large (e.g. $2^{32}$)
  - ???

Fixing impracticality: RadixSort
- Radix = “The base of a number system”
  - We’ll use 10 for convenience, but could be anything
- Idea: BucketSort on each digit, least significant to most significant (lsd to msd)

Radix Sort Example (1st pass)
Input data
3
9
123
537
67
478
38
9

Bucket sort by 1's digit
After 1st pass

Radix Sort Example (2nd pass)
After 1st pass

Bucket sort by 10's digit
After 2nd pass

Radix Sort Example (3rd pass)
After 2nd pass

Bucket sort by 100's digit
After 3rd pass

Invariant: after $k$ passes the low order $k$ digits are sorted.
Radixsort: Complexity

• How many passes?
• How much work per pass?
• Total time?
• Conclusion?
• In practice
  – RadixSort only good for large number of elements with relatively small values
  – Hard on the cache compared to MergeSort/QuickSort

Internal versus External Sorting

• Need sorting algorithms that minimize disk/tape access time
• External sorting – Basic Idea:
  – Load chunk of data into RAM, sort, store this “run” on disk/tape
  – Use the Merge routine from Mergesort to merge runs
  – Repeat until you have only one run (one sorted chunk)
  – Text gives some examples

Graphs

Chapter 9 in Weiss

Graph Definitions

In directed graphs, edges have a specific direction:

In undirected graphs, they don’t (edges are two-way):

Graph... ADT?

• Not quite an ADT...
  operations not clear
• A formalism for representing relationships between objects
  Graph \( G = (V, E) \)
  – Set of vertices:
    \( V = \{v_1, v_2, \ldots, v_n\} \)
  – Set of edges:
    \( E = \{e_1, e_2, \ldots, e_m\} \)
    where each \( e_i \) connects two vertices \( (v_{i_1}, v_{i_2}) \)

More Definitions: Simple Paths and Cycles

A simple path repeats no vertices (except that the first can be the last):

- \( p = \{\text{Seattle, Salt Lake City, San Francisco, Dallas}\} \)
- \( p = \{\text{Seattle, Salt Lake City, Dallas, San Francisco, Seattle}\} \)

A cycle is a path that starts and ends at the same node:

- \( p = \{\text{Seattle, Salt Lake City, Seattle, San Francisco, Seattle}\} \)

A simple cycle is a cycle that repeats no vertices except that the first vertex is also the last (in undirected graphs, no edge can be repeated)
Trees as Graphs

- Every tree is a graph!
- Not all graphs are trees!

A graph is a tree if
- There are no cycles (directed or undirected)
- There is a path from the root to every node

Directed Acyclic Graphs (DAGs)

DAGs are directed graphs with no (directed) cycles.

Aside: If program call-graph is a DAG, then all procedure calls can be inlined

Graph Representations

0. List of vertices + list of edges
1. 2-D matrix of vertices (marking edges in the cells) “adjacency matrix”
2. List of vertices each with a list of adjacent vertices “adjacency list”

Things we might want to do:
- iterate over vertices
- iterate over edges
- iterate over vertices adj. to a vertex
- check whether an edge exists

Representation 1: Adjacency Matrix

A $|V| \times |V|$ array in which an element $(u, v)$ is true if and only if there is an edge from $u$ to $v$

Space requirements: runtime:

Representation

- adjacency matrix:

$$A[u][v] = \begin{cases} \text{weight}, & \text{if } (u, v) \in E \\ 0, & \text{if } (u, v) \notin E \end{cases}$$

- adjacency list:
Representation 2: Adjacency List
A $|V|$-ary list (array) in which each entry stores a list (linked list) of all adjacent vertices.

Some Applications: Moving Around Washington
What’s the shortest way to get from Seattle to Pullman?

Some Applications: Reliability of Communication
If Wenatchee’s phone exchange goes down, can Seattle still talk to Pullman?

Some Applications: Bus Routes in Downtown Seattle
If we’re at 3rd and Pine, how can we get to 1st and University using Metro?

Application: Topological Sort
Given a directed graph, $G = (V, E)$, output all the vertices in $V$ such that no vertex is output before any other vertex with an edge to it.

Is the output unique?
Topological Sort: Take One

1. Label each vertex with its in-degree (# of inbound edges)
2. While there are vertices remaining:
   a. Choose a vertex \(v\) of in-degree zero; output \(v\)
   b. Reduce the in-degree of all vertices adjacent to \(v\)
   c. Remove \(v\) from the list of vertices

Runtime:

```cpp
void Graph::topsort()
{
    Vertex v, w;
    labelEachVertexWithItsIn-degree();
    for (int counter=0; counter < NUM_VERTICES; counter++)
    {
        v = findNewVertexOfDegreeZero();
        v.topologicalNum = counter;
        for each w adjacent to v
            w.indegree--;
    }
}
```