CS180 Discussion

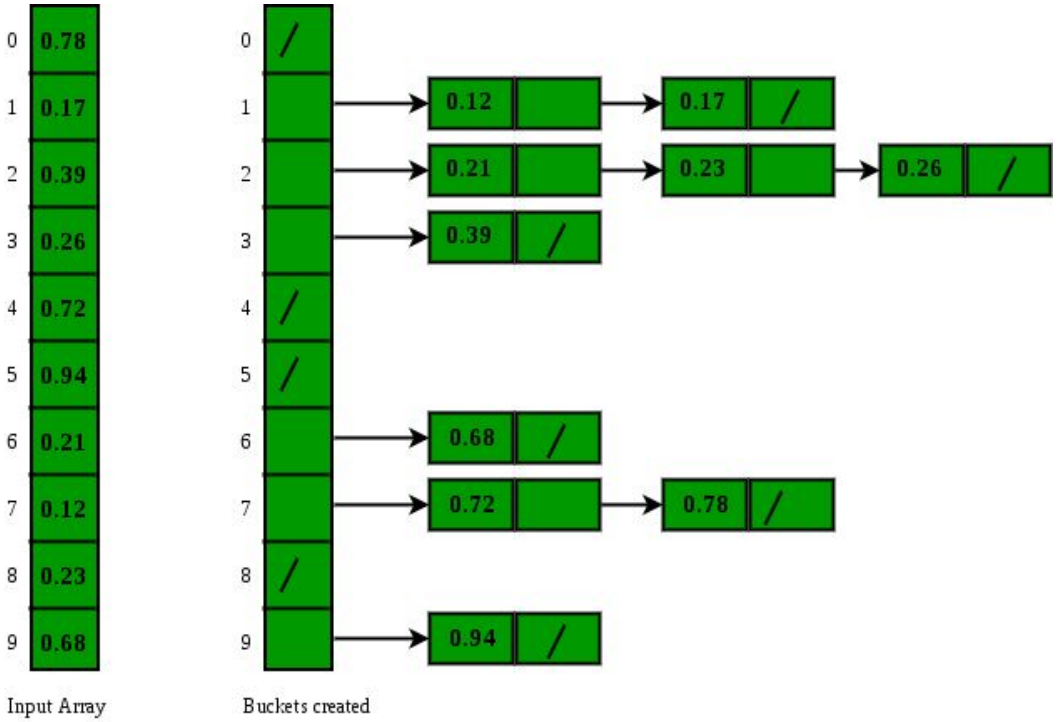
Week 8

Lecture Recap

- Bucket sort
- Radix sort
- Network flow
    - Ford Fulkerson
    - Bipartite matching

Bucket Sort

1. Create n empty buckets
2. Do following for every array element arr[i]
   a. Insert arr[i] into bucket[n*array[i]]
   b. Sort individual buckets using insertion sort.
3. Concatenate all sorted buckets.

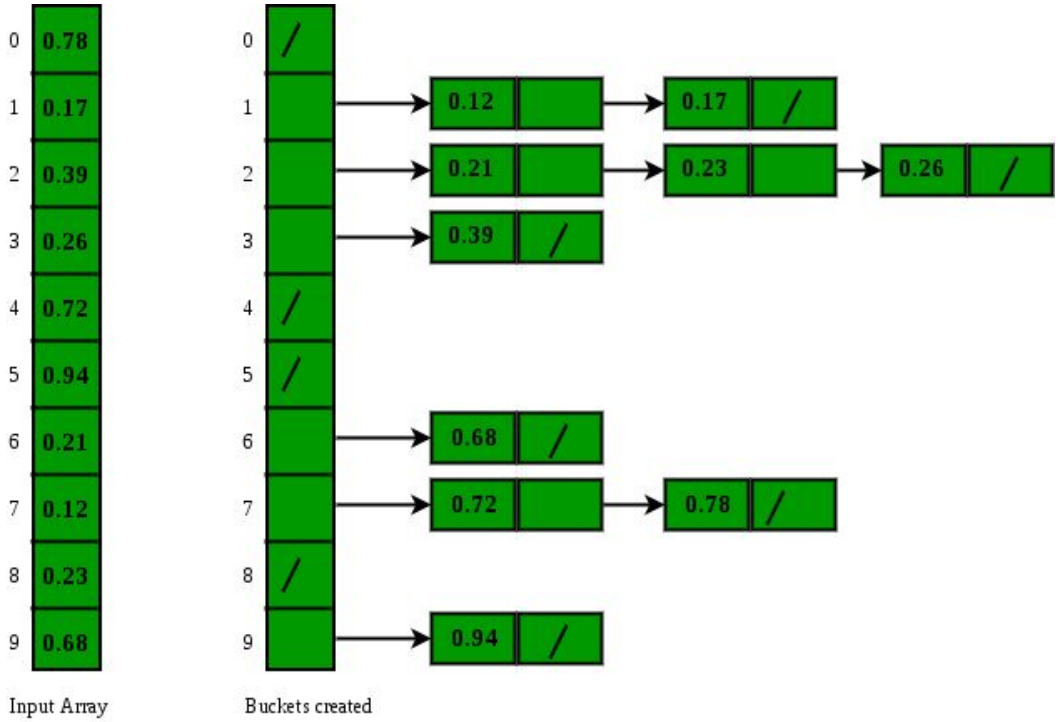


Makes sense for uniformly distributed elements

Bucket Sort

1. Create n empty buckets
2. Do following for every array element arr[i]
   a. Insert arr[i] into bucket[n*array[i]]
   b. Sort individual buckets using insertion sort.
3. Concatenate all sorted buckets.

O(n + k)



Radix Sort

Original, unsorted list:

Makes sense when elements go to n^2

170, 45, 75, 90, 802, 24, 2, 66

- Sorting by least significant digit (1s place) gives:
  - 170, 90, 802, 2, 24, 45, 75, 66
- Sorting by next digit (10s place) gives:
  - 802, 2, 24, 45, 66, 170, 75, 90
- Sorting by the most significant digit (100s place) gives:
  - 2, 24, 45, 66, 75, 90, 170, 802

Radix Sort

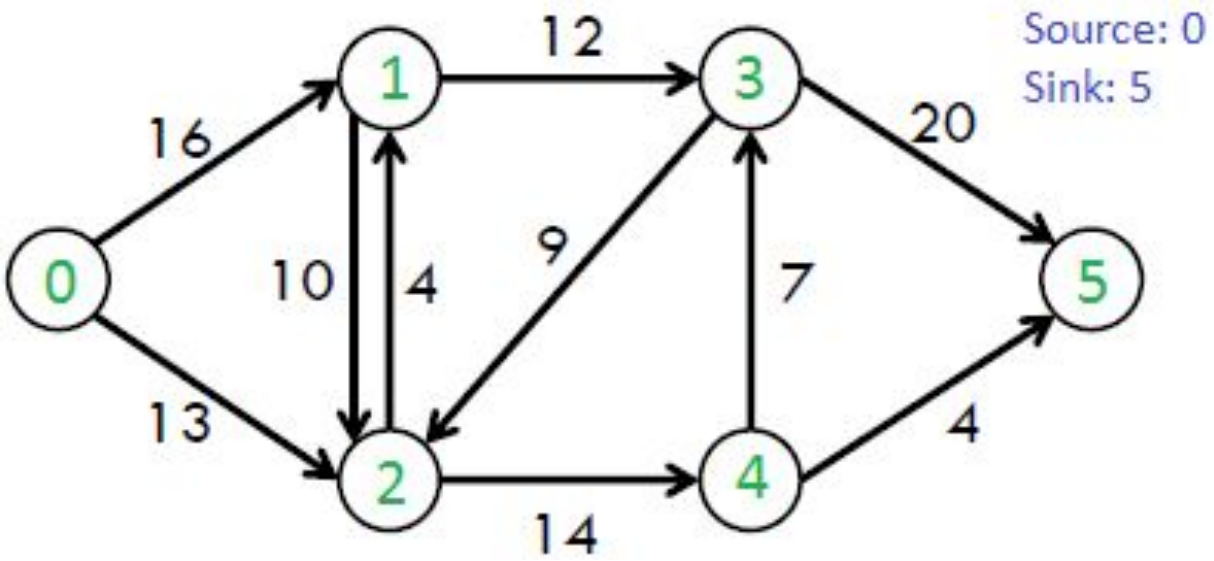
Original, unsorted list:

$O((n+base) * \log_b(max))$

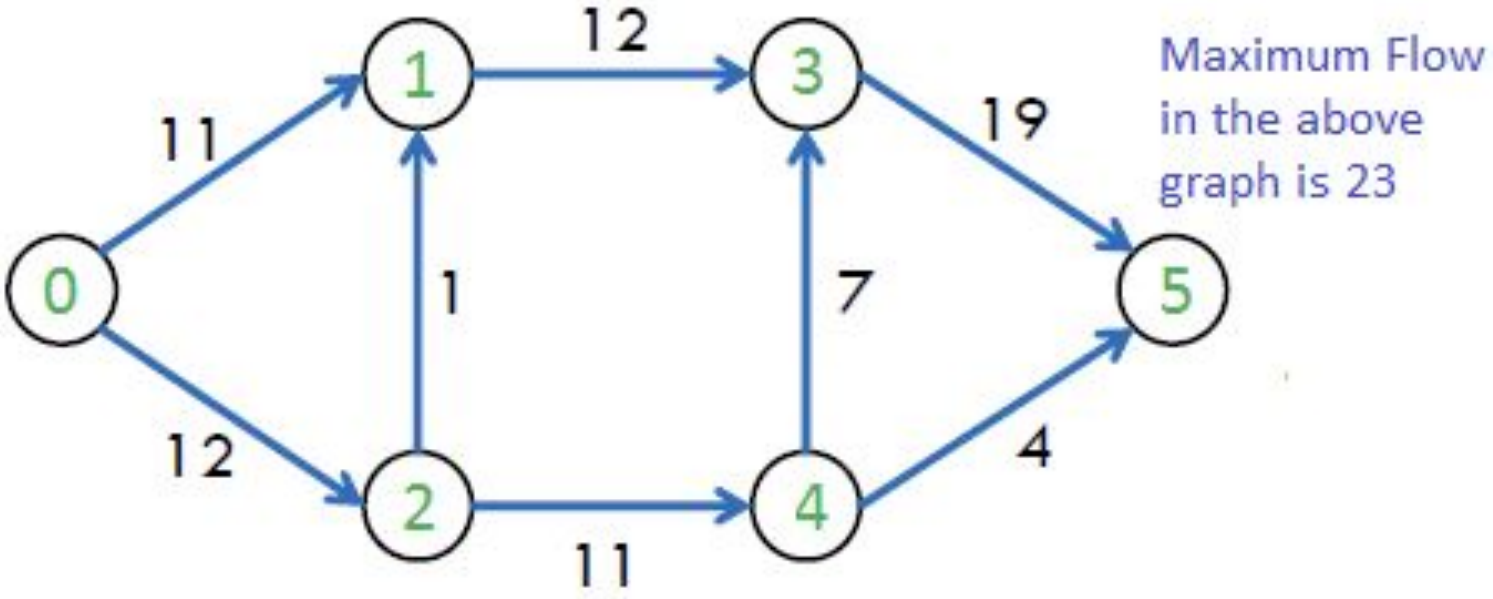
170, 45, 75, 90, 802, 24, 2, 66

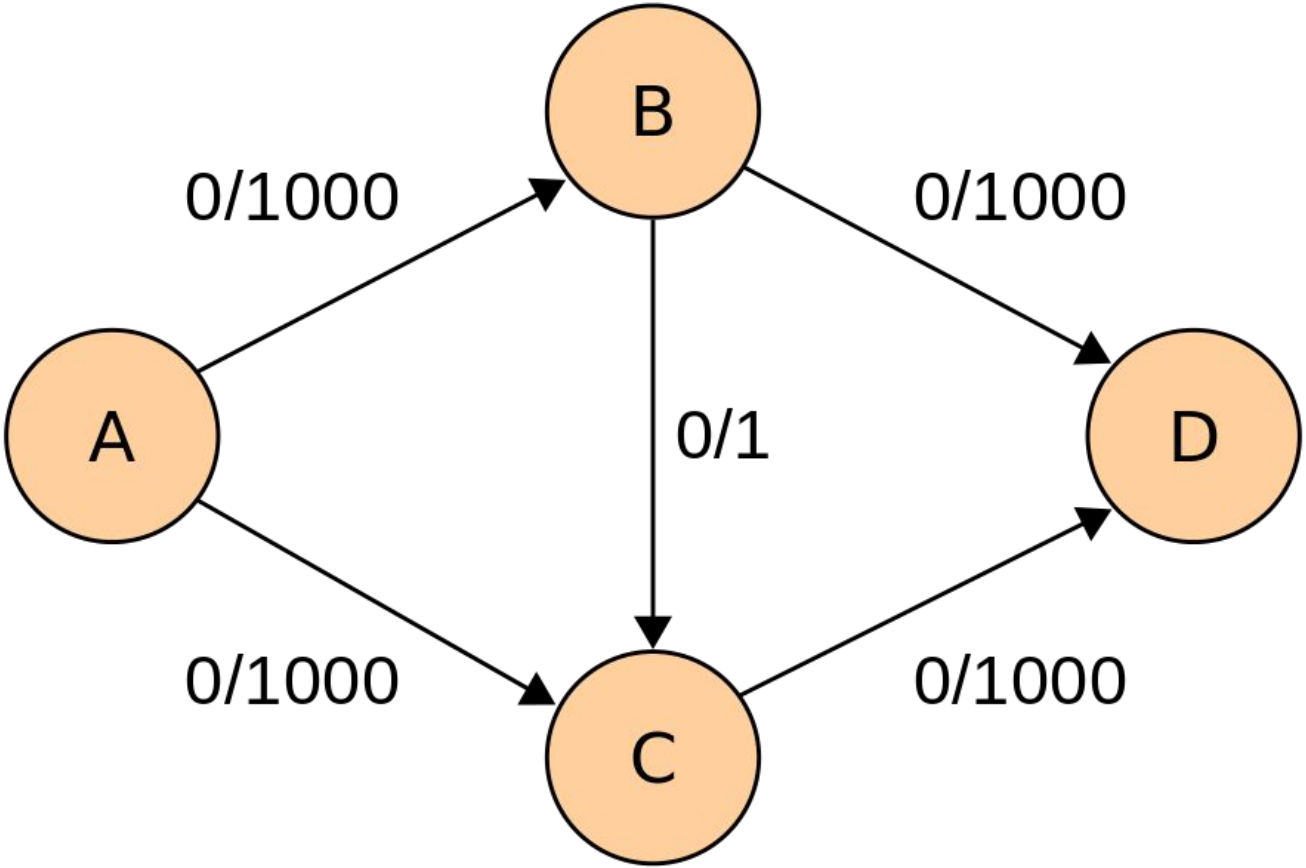
- Sorting by least significant digit (1s place) gives:
  - 170, 90, 802, 2, 24, 45, 75, 66
- Sorting by next digit (10s place) gives:
  - 802, 2, 24, 45, 66, 170, 75, 90
- Sorting by the most significant digit (100s place) gives:
  - 2, 24, 45, 66, 75, 90, 170, 802

Ford Fulkerson



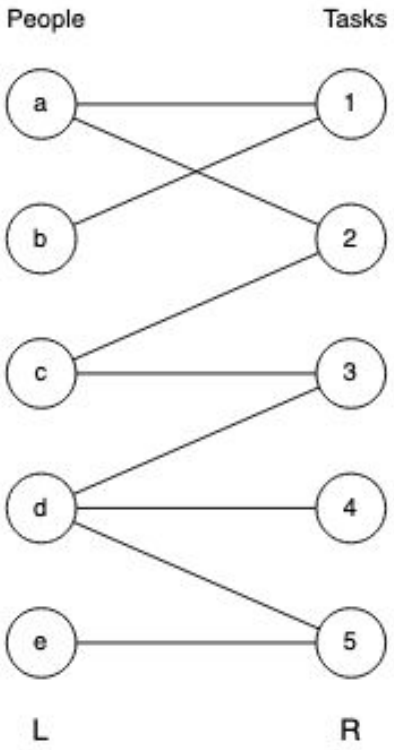
Ford Fulkerson



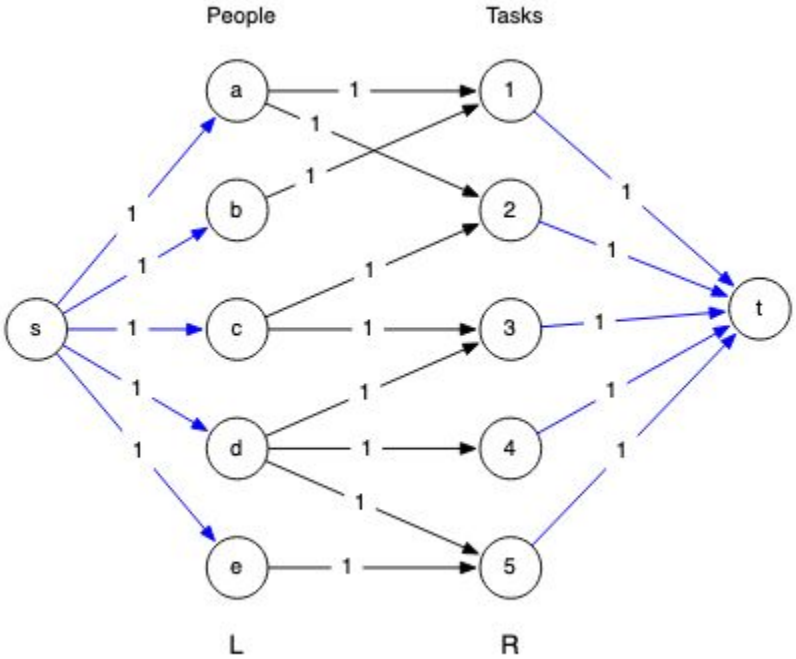

B
0/1000
0/1000
A
0/1
D
0/1000
0/1000
C

Reducing Matching to Net Flow

- Suppose we have a set of people L and set of jobs R.
- Each person can do only some of the jobs.
- Can model this as a bipartite graph $\rightarrow$



Reducing Matching to Net Flow



Reducing Matching to Net Flow

Need to show:

- If there is a matching of k edges, there is a flow f of value k.
  - Graph has 1 unit of flow across each of the k edges
  - 1 unit leaves & enters each node (except s,t)

- If there is a flow f of value k, there is a matching with k edges.
  - We find the maximum flow f (say with k edges)
  - This corresponds to a pairing M of k edges, since max flow is 1 between edges
  - If there were a matching with>k edges, we would have found a flow with value>k, contradicting that f was maximum
  - Hence,M is maximum

Unique minimum cut

Let G=(V,E) be a directed graph, with source $s \in V$, sink $t \in V$, and nonnegative edge capacities {ce}. Give a polynomial-time algorithm to decide whether G has a unique minimum s-t cut (i.e., an s-t of capacity strictly less than that of all other s-t cuts)

Unique minimum cut

1) Run Ford-Fulkerson algorithm and consider the final residual graph.

2) Find the set of vertices that are reachable from the source in the residual graph.

3) All edges which are from a reachable vertex to non-reachable vertex are minimum cut edges. Print all such edges.

Interview Questions!!!

Interview Questions!!!

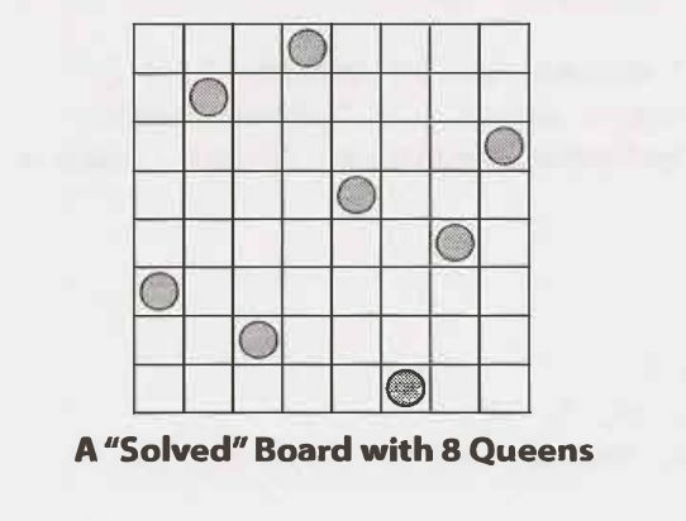
Interview Questions!!!

Interview Questions!!!

Interview Questions!!!

Eight queens

Write an algorithm to print all ways of arranging eight queens on an 8x8 chess board so that none of them share the same row, column, or diagonal. In this case, "diagonal" means all diagonals, not just the two that bisect the board.



A "Solved" Board with 8 Queens

Eight queens

```
1  int GRID_SIZE = 8;
2
3  void placeQueens(int row, Integer[] columns, ArrayList<Integer[]> results) {
4    if (row == GRID_SIZE) { // Found valid placement
5      results.add(columns.clone());
6    } else {
7      for (int col = 0; col < GRID_SIZE; col++) {
8        if (checkValid(columns, row, col)) {
9          columns[row] = col;  // Place queen
10         placeQueens(row + 1, columns, results);
11       }
12     }
13   }
14 }
15
16 /* Check if (row1, column1) is a valid spot for a queen by checking if there is a
17  * queen in the same column or diagonal. We don't need to check it for queens in
18  * the same row because the calling placeQueen only attempts to place one queen at
19  * a time. We know this row is empty. */
20 boolean checkValid(Integer[] columns, int row1, int column1) {
21   for (int row2 = 0; row2 < row1; row2++) {
22     int column2 = columns[row2];
23     /* Check if (row2, column2) invalidates (row1, column1) as a
24      * queen spot. */
25
26     /* Check if rows have a queen in the same column */
27     if (column1 == column2) {
28       return false;
29     }
30
31     /* Check diagonals: if the distance between the columns equals the distance
32      * between the rows, then they're in the same diagonal. */
33     int columnDistance = Math.abs(column2 - column1);
34
35     /* row1 > row2, so no need for abs */
36     int rowDistance = row1 - row2;
37     if (columnDistance == rowDistance) {
38       return false;
39     }
40   }
41   return true;
42 }
```