Exhaustive search as a baseline for algorithm design

Summary

Exhaustive Search

- Estimate the number of candidates
- Generate candidates one at a time and test for the optimal solution

Optimization techniques for Exhaustive Computation

1. Avoid recomputation between successive candidates (Max-sublist 2, KMP)
2. Reduce the size of the candidate set (Max-sublist 3, Euclidean GCD)
3. Eliminate non-promising candidates during the search: backtracking (n-Queens problem)

Exhaustive algorithms: Sorting

Selection sort

- Scan array to find smallest element
- Scan array to find second smallest element
- etc.

Complexity?

Can we do better?

Exhaustive algorithms: Sorting

Selection sort

- Scan array to find smallest element
- Scan array to find second smallest element
- etc.

  Complexity? $O(n^2)$

Can we do better? Yes. See *divide-and-conquer*.

Exhaustive algorithms: Searching

Sequential scan:

- Go through the entire list of *n* items to find the desired item

  Complexity?

Can we do better?

Exhaustive algorithms: Searching

Sequential scan:

- Go through the entire list of *n* items to find the desired item

  Complexity? O(n)

Can we do better?

No. Not really.

Exhaustive algorithms: graph traversals

DFS and BFS:

- Shortest paths in unweighted graphs
- Topological sorting
- Discovering strongly-connected components

  Complexity?

Can we do better?

Exhaustive algorithms: graph traversals

DFS and BFS:

- Shortest paths in unweighted graphs
- Topological sorting
- Discovering strongly-connected components

  Complexity? $O(n + m)$

Can we do better?

No. Not really. We have to traverse all the vertices and edges

Knapsack 01 (discrete items)

Exhaustive knapsack algorithm for n items:

- Generate all possible knapsacks
- Discard all combinations that do not fit
- Compute value of each knapsack and select thje one with max value

  Complexity?

Can we do better?

Knapsack 01 (discrete items)

Exhaustive knapsack algorithm for n items:

- Generate all possible knapsacks
- Discard all combinations that do not fit
- Compute value of each knapsack and select thje one with max value

  Complexity? $O(2^n)$

Can we do better?

Yes. See *dynamic programming*

Introducing Closest Pair

Closest-Pair Problem

Input: *n* points in *d*-dimensional space
Output: a pair of points with the smallest distance between them

Motivation

- Airplanes close to colliding
- Which post offices should be closed
- Which DNA sequences are most similar
- The nearest-neighbor classifier

Brute Force for Closest Pair

- Exhaustive Solution (for 2-D case):
  - Compute distances between all pairs of points

    $\text{sqrt}((x_i - x_j)^2 + (y_i - y_j)^2)$
  - Scan all distances to find smallest
- Complexity: $O(n^2)$, assuming each numerical operation is constant time (including square root?)
- Improvements:
  - Drop the square root
  - Don't compute distance for same 2 points twice
  - Does this improve complexity?

Can we do better?

Yes, see *divide-and-conquer*.

Summary of algorithms so far

- Graph Traversals
- GCD*
- Generating primes*
- Max sublist*
- Sorting*: selection sort
- Searching: pattern search*
- Geometry: the closest pair*
- Knapsack 01*

---

* Can be improved just by applying an optimization ...

* Can be improved with *divide-and-conquer*

* Can be improved with *dynamic programming*