CSE 421

Union Find DS
Dijkstra’s Algorithm,

Shayan Oveis Gharan
Kruskal's Algorithm [1956]

Kruskal(G, c) {
    Sort edges weights so that \( c_1 \leq c_2 \leq \ldots \leq c_m \).
    \( T \leftarrow \emptyset \)

    foreach \((u \in V)\) make a set containing singleton \(\{u\}\)

    for \(i = 1\) to \(m\)
        Let \((u,v) = e_i\)
        if (u and v are in different sets) {
            \(T \leftarrow T \cup \{e_i\}\)
            merge the sets containing \(u\) and \(v\)
        }
    return \(T\)
}

Kruskal's Algorithm [1956]
Union Find Data Structure

Each set is represented as a tree of pointers, where every vertex is labeled with longest path ending at the vertex.

To check whether A,Q are in same connected component, follow pointers and check if root is the same.
Union Find Data Structure

**Merge**: To merge two connected components, make the root with the smaller label point to the root with the bigger label (adjusting labels if necessary). Runs in $O(1)$ time

At most one label must be adjusted.
Kruskal’s Algorithm with Union Find

Implementation. Use the union-find data structure.

- Build set $T$ of edges in the MST.
- Maintain a set for each connected component.
- $O(m \log n)$ for sorting and $O(m \log n)$ for union-find

```
Kruskal(G, c) {
    Sort edges weights so that $c_1 \leq c_2 \leq \ldots \leq c_m$.
    $T \leftarrow \emptyset$

    foreach $(u \in V)$ make a set containing singleton $\{u\}$

    for $i = 1$ to $m$
        Let $(u,v) = e_i$
        if (u and v are in different sets) {
            $T \leftarrow T \cup \{e_i\}$
            merge the sets containing $u$ and $v$
        }

    return $T$
}
```
Claim: If the label of a root is $k$, there are at least $2^k$ elements in the set.

Therefore the depth of any tree in algorithm is at most $\log n$

So, we can check if $u, v$ are in the same component in time $O(\log n)$
Claim: If the label of a root is k, there are at least $2^k$ elements in the set.

Pf: By induction on k.

Base Case (k = 0): this is true. The set has size 1.

IH: Suppose the claim is true until some time t

IS: If we merge roots with labels $k_1 > k_2$, the number of vertices only increases while the label stays the same. If $k_1 = k_2$, the merged tree has label $k_1 + 1$, and by induction, it has at least

$$2^{k_1} + 2^{k_2} = 2^{k_1+1}$$

elements.
Suppose edge weights are not distinct, and Kruskal’s algorithm sorts edges so
\[ c_{e_1} \leq c_{e_2} \leq \cdots \leq c_{e_m} \]

Suppose Kruskal finds tree \( T \) of weight \( c(T) \), but the optimal solution \( T^* \) has cost \( c(T^*) < c(T) \).

Perturb each of the weights by a very small amount so that
\[ c'_{e_1} < c'_{e_2} < \cdots < c'_{e_m} \]

where \( c'_{e_i} = c_{e_i} + i \cdot \epsilon \)

If \( \epsilon \) is small enough, \( c'(T^*) < c(T) \).

However, this contradicts the correctness of Kruskal’s algorithm, since the algorithm will still find \( T \), and Kruskal’s algorithm is correct if all weights are distinct.
Summary (Greedy Algorithms)

- **Greedy Stays Ahead**: Interval Scheduling
- **Structural**: Interval Partitioning
- **Exchange Arguments**: MST, Kruskal’s Algorithm,
- **Data Structures**: Union Find
Divide and Conquer Approach
Divide and Conquer

Similar to algorithm design by induction, we reduce a problem to several subproblems. Typically, each sub-problem is at most a constant fraction of the size of the original problem.

Recursively solve each subproblem
Merge the solutions

Examples:
- Mergesort, Binary Search, Strassen’s Algorithm,
A Classical Example: Merge Sort

A

Split to n/2

merge

sort recursively
Why Balanced Partitioning?

An alternative "divide & conquer" algorithm:

- Split into $n-1$ and 1
- Sort each sub problem
- Merge them

Runtime

$$T(n) = T(n - 1) + T(1) + n$$

Solution:

$$T(n) = n + T(n - 1) + T(1)$$

$$= n + n - 1 + T(n - 2)$$

$$= n + n - 1 + n - 2 + T(n - 3)$$

$$= n + n - 1 + n - 2 + ⋯ + 1 = O(n^2)$$
D&C: The Key Idea

Suppose we've already invented Bubble-Sort, and we know it takes $n^2$

Try just one level of divide & conquer:

Bubble-Sort(first n/2 elements)
Bubble-Sort(last n/2 elements)

Merge results

Time: $2T(n/2) + n = n^2/2 + n \ll n^2$

Almost twice as fast!
D&C approach

• “the more dividing and conquering, the better”
  • Two levels of D&C would be almost 4 times faster, 3 levels almost 8, etc., even though overhead is growing.
  • Best is usually full recursion down to a small constant size (balancing "work" vs "overhead").
    In the limit: you’ve just rediscovered mergesort!

• Even unbalanced partitioning is good, but less good
  • Bubble-sort improved with a 0.1/0.9 split:
    \[(.1n)^2 + (.9n)^2 + n = .82n^2 + n\]
    The 18% savings compounds significantly if you carry recursion to more levels, actually giving \(O(n \log n)\), but with a bigger constant.

• This is why Quicksort with random splitter is good – badly unbalanced splits are rare, and not instantly fatal.
Finding the Root of a Function
Finding the Root of a Function

Given a continuous function $f$ and two points $a < b$ such that

\[
\begin{align*}
    f(a) &\leq 0 \\
    f(b) &\geq 0
\end{align*}
\]

Find an approximate root of $f$ (a point $c$ where there is $r$ s.t., $|r - c| \leq \epsilon$ and $f(r) = 0$).

Note $f$ has a root in $[a, b]$ by intermediate value theorem.

Note that roots of $f$ may be irrational.

So, we want to approximate the root with an arbitrary precision!

\[
f(x) = \sin(x) - \frac{100}{\sqrt{x}} + x^4
\]
A Naive Approach

Suppose we want $\epsilon$ approximation to a root.

Divide $[a, b]$ into $n = \frac{b-a}{\epsilon}$ intervals. For each interval check $f(x) \leq 0, f(x + \epsilon) \geq 0$

This runs in time $O(n) = O\left(\frac{b-a}{\epsilon}\right)$

Can we do faster?
D&C Approach (Based on Binary Search)

\[ \text{Bisection}(a,b, \varepsilon) \]

\[
\begin{align*}
&\text{if } (b - a) < \varepsilon \text{ then} \\
&\quad \text{return } (a) \\
&\text{else} \\
&\quad m \leftarrow (a + b)/2 \\
&\quad \text{if } f(m) \leq 0 \text{ then} \\
&\quad\quad \text{return } \text{Bisection}(c, b, \varepsilon) \\
&\quad\text{else} \\
&\quad\quad \text{return } \text{Bisection}(a, c, \varepsilon)
\end{align*}
\]
Let \( n = \frac{a-b}{\epsilon} \)

And \( c = (a + b)/2 \)

Always half of the intervals lie to the left and half lie to the right of \( c \)

So,

\[
T(n) = T\left(\frac{n}{2}\right) + O(1)
\]

i.e., \( T(n) = O(\log n) = O\left(\log \frac{a-b}{\epsilon}\right) \)
Correctness Proof

\[ P(k) = \text{"For any } a, b \text{ such that } k \epsilon \leq |a - b| \leq (k + 1) \epsilon \text{ if } f(a)f(b) \leq 0, \text{ then we find an } \epsilon \text{ approx to a root using } \log k \text{ queries to } f" \]

**Base Case:** \( P(1) \): Output \( a + \epsilon \)

**IH:** Assume \( P(k) \).

**IS:** Show \( P(2k) \). Consider an arbitrary \( a, b \) s.t.,
\[ 2k \epsilon \leq |a - b| < (2k + 1) \epsilon \]

If \( f(a + k \epsilon) = 0 \) output \( a + k \epsilon \).

If \( f(a)f(a + k \epsilon) < 0 \), solve for interval \( a, a + k \epsilon \) using \( \log(k) \) queries to \( f \).

Otherwise, we must have \( f(b)f(a + k \epsilon) < 0 \) since \( f(a)f(b) < 0 \) and \( f(a)f(a + k \epsilon) \geq 0 \). Solve for interval \( a + k \epsilon, b \).

Overall we use at most \( \log(k) + 1 = \log(2k) \) queries to \( f \).
Finding the Closest Pair of Points
Closest Pair of Points (non geometric)

Given $n$ points and arbitrary distances between them, find the closest pair. (E.g., think of distance as airfare – definitely not Euclidean distance!)

Must look at all $n$ choose 2 pairwise distances, else any one you didn’t check might be the shortest. i.e., you have to read the whole input
Closest Pair of Points (1-dimension)

Given $n$ points on the real line, find the closest pair, e.g., given 11, 2, 4, 19, 4.8, 7, 8.2, 16, 11.5, 13, 1 find the closest pair

Fact: Closest pair is adjacent in ordered list
So, first sort, then scan adjacent pairs.

Key point: do not need to calc distances between all pairs: exploit geometry + ordering
Closest Pair of Points (2-dimensions)

Given n points in the plane, find a pair with smallest Euclidean distance between them.

Fundamental geometric primitive.
  Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
  Special case of nearest neighbor, Euclidean MST, Voronoi.

Brute force: Check all pairs of points p and q with $\Theta(n^2)$ time.

Assumption: No two points have same x or y coordinates.
A Divide and Conquer Alg

**Divide:** draw vertical line $L$ with $\approx n/2$ points on each side.

**Conquer:** find closest pair on each side, recursively.

**Combine** to find closest pair overall

Return best solutions

$12 \rightarrow 8 \rightarrow 21$ seems like $\Theta(n^2)$?