CSE 421

Dijkstra’s Algorithm, Divide and Conquer

Shayan Oveis Gharan
Boiling Water Example

Q: Given an empty bowl, how do you make boiling water?

A: Well, I fill it with water, turn on the stove, leave the bowl on the stove for 20 minutes. I have my boiling water.

Q: Now, suppose you have a bowl of water, how do you make boiling water?

A: First, I pour water away, now I have an empty bowl and I have already solved this!
Lesson: Never solve a problem twice!

Related Q: Try to do max-spanning tree using min-spanning tree
Dijkstra’s Algorithm: Example

Assum $c_e > 0$
Dijkstra’s Algorithm: Example

![Dijkstra's Algorithm Example Graph](image-url)
Dijkstra’s Algorithm: Example

Graph:
- **s** (source) connected to:
  - **0** (current node) with weight 2
  - **2** with weight 7
  - **4** with weight 5
- **2** (current node) connected to:
  - **0** (current node) with weight 2
  - **4** with weight 4
- **4** connected to:
  - **2** (current node) with weight 4
  - **8** with weight 3
  - **5** with weight 1

Path:
- Initial path: s → 0 → 2
- Minimum weight: 2 + 4 = 6
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example

Graph representation of Dijkstra’s Algorithm.
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example

![Graph Diagram]
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Dijkstra’s Algorithm: Example
Disjkstra’s Algorithm: Correctness

Prove by induction that throughout the algorithm, for any \( u \in S \), the path \( P_u \) in the shortest from \( s \) to \( u \).

**Base Case:** This is always true when \( S = \{s\} \).

**IH:** Suppose \( |S| = k \) and the claim holds for \( S \).

**IS:** Say \( v \) is the \( k+1 \)-st vertex that we add to \( S \). Let \( \{u,v\} \) be last edge on \( P_v \).

If \( P_v \) is not the shortest path there is a path \( P \) to \( s \) which is shorter.

Consider the **first** time that \( P \) leaves \( S \) (with edge \( \{x,y\} \)).

\( S \rightarrow x \) has weight (at least) \( d(x) \)

So, \( c(P) \geq d(x) + c_{x,y} \geq d(v) = c(P_v) \).

A contradiction.
Remarks on Dijkstra’s Algorithm

- Algorithm also produces a tree of shortest paths to s following Parent links
- Algorithm works on directed graph (with nonnegative weights)
- The algorithm fails with negative edge weights.
  - e.g., some airline tickets

Why does it fail?

- Dijkstra’s algorithm is similar to BFS:
  - Substitute every edge with $c_e = k$ with a path of length $k$, then run BFS.

\[
d_{\text{Dijkstra}} = \text{BFS}.
\]
Implementing Dijkstra’s Algorithm

**Priority Queue:** Elements each with an associated key

- **Operations**
  - **Insert**
  - **Find-min**
    - Return the element with the smallest key
  - **Delete-min**
    - Return the element with the smallest key and delete it from the data structure
  - **Decrease-key**
    - Decrease the key value of some element

**Implementations**

**Arrays:**
- **O(n)** time find/delete-min,
- **O(1)** time insert/decrease key

**Binary Heaps:**
- **O(log n)** time insert/decrease-key/delete-min,
- **O(1)** time find-min
Dijkstra’s Algorithm

Runs in $O((n+m)\log n)$.

```plaintext
Dijkstra(G, c, s) {
    foreach (v ∈ V) d[v] ← ∞  //This is the key of node v
    d[s] ← 0
    foreach (v ∈ V) insert v onto a priority queue Q
    Initialize set of explored nodes S ← {s}

    while (Q is not empty) {  $O(n)$
        u ← delete min element from Q
        S ← S ∪ { u }
        foreach (edge e = (u, v) incident to u)
            if ((v ∉ S) and (d[u]+c_e < d[v]))
                d[v] ← d[u] + c_e
                Decrease key of v to d[v].
                Parent(v) ← u
    }
}
```

$O(m)$ of decrease key, each runs in $O(\log n)$

$O(n)$ of delete min, each in $O(\log n)$
Summary (Greedy Algorithms)

- **Greedy Stays Ahead**: Interval Scheduling, Dijkstra’s algorithm
- **Structural**: Interval Partitioning
- **Exchange Arguments**: MST, Kruskal’s Algorithm,
- **Data Structures**: Union Find, Heap
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

Split to n/2

sort recursively

merge
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:

\[ T(n) = n + T(n - 1) + T(1) \]
\[ = n + n - 1 + T(n - 2) \]
\[ = n + n - 1 + n - 2 + T(n - 3) \]
\[ = n + n - 1 + n - 2 + \cdots + 1 = O(n^2) \]
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 $\frac{n}{2}$ elements)
- Bubble-Sort(last $\frac{n}{2}$ elements)

Merge results

Time: $2 T(\frac{n}{2}) + n = \frac{n^2}{2} + n \ll n^2$

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

\[
\begin{cases}
    f(a) \leq 0 \\
    f(b) \geq 0
\end{cases}
\]

Find an approximate root of $f$ (a point $c$ where $f(c) = 0$).

$f$ 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)

**Bisection**(a, b, ε)

if $(b - a) < \epsilon$ then

    return (a)

else

    $m \leftarrow (a + b)/2$

    if $f(m) \leq 0$ then

        return(Bisection(c, b, ε))

    else

        return(Bisection(a, c, ε))

\[ \text{Diagram showing the bisection method with } a, c, b, \text{ and the midpoint } m. \]
Let $n = \frac{a-b}{\varepsilon}$

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., $T(n) = O(\log n) = O(\log \frac{a-b}{\varepsilon})$
Correctness Proof

P(k) = “For any \(a, b\) such that \(k\varepsilon \leq |a - b| \leq (k + 1)\varepsilon\) if \(f(a)f(b) < 0\), then we find an \(\varepsilon\) approx to a root using \(\log k\) queries to \(f\)”

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

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

**IS:** Show P(2k). Consider an arbitrary \(a, b\) s.t.,
\[
2k\varepsilon \leq |a - b| < (2k + 1)\varepsilon
\]
If \(f(a + k\varepsilon) = 0\) output \(a + k\varepsilon\).
If \(f(a)f(a + k\varepsilon) < 0\), solve for interval \(a, a + k\varepsilon\) using \(\log(k)\) queries to \(f\).
Otherwise, we must have \(f(b)f(a + k\varepsilon) < 0\) since \(f(a)f(b) < 0\) and \(f(a)f(a + k\varepsilon) \geq 0\). Solve for interval \(a + k\varepsilon, b\).
Overall we use at most \(\log(k) + 1 = \log(2k)\) queries to \(f\).