Deterministic Algorithms

- So far: deterministic algorithms on worst case inputs.
- Why deterministic algorithms?
  - Easier to understand, pretty powerful.
- Why worst case?
  - Enables precise statements
  - But maybe not reflective of real-world instances.
  - Average-case analysis? What distribution?

Why Randomized Algorithms?

- Efficient in expectation
- Optimal with high probability
- Break (undesired) symmetry
- Show some solution exists, or derive bound on their number

Types of randomized algorithms:

- Fail with some small probability
- Always succeed, but running time may be non-polynomial

Examples

- Min-Cut
- Randomized Median Finding
- Max 3-SAT

Minimum Cuts

Problem. Given undirected $G = (V, E)$, partition $V$ into sets $A, V \setminus A$, minimizing edge count across cut:

$$\text{cut}(A) = |\{(u, v) \in E, u \in A, v \notin A\}|$$

- We saw how to compute minimum $s - t$ cut in directed graph.
- for fixed $s, t$
- How do we compute global minimum cut? Is this harder?

Deterministic Algorithm

Idea. Convert into $s - t$ cut in directed graph.
Replace edge $e = (u, v)$ with directed edges both ways (capacity 1).
Pick arbitrary $s$.
for each other vertex $t$
Compute minimum $s - t$ cut.
Return smallest computed $s - t$ cut.

Running Time. $n$ max-flow computations $\Rightarrow O(mn^2)$ at best.

Contraction Algorithm (Karger, 1995)

- Idea: only edges across cut matter, edges within set don’t
- Collapse $S$ and $V \setminus S$ into one node each
- But allow multiple edges between nodes (multigraph)

Def. Multigraph $G = (V, E)$ is a graph that can have parallel edges.
Def. Contracting an edge $(u, v)$ in $G = (V, E)$ produces a new multigraph $G' = (V', E')$

- With new node $w$ instead of $u, v$ (edges $(u, v)$ deleted).
- If $(x, u)$ or $(x, v) \in E$, then $(x, w) \in E'$.
- All other edges preserved.
Contraction Algorithm

- Pick edge \((u, v) \in E\) uniformly at random
- Contract edge \((u, v)\), replacing \(u, v\) with new node \(w\)
- Preserve edges, setting \(u\) and \(v\) endpoints to \(w\)
- Keep multiple edges, delete self-loops

\[ S(v) = \{v\} \text{ for all } v \in V. \]

\textbf{while } |V| > 2 \textbf{ do}

\begin{itemize}
  \item Pick edge \((u, v) \in E\) uniformly at random.
  \item Contract edge \((u, v)\) to get \(G'\) with new node \(w\).
  \item Set \(S(w) \leftarrow S(u) \cup S(v)\).
  \item Update \(G \leftarrow G'\).
\end{itemize}

\textbf{return } \(S(v)\) for \(v \in V\).

\textbf{Contraction Algorithm Analysis}

\textbf{Theorem.} Alg finds global min cut with probability at least \(1/\binom{n}{2}\).

\textbf{Proof.} Suppose \((A, B)\) is a global min cut with \(\text{cut}(A, B) = k\).

- What could go wrong in the first step?
- Select \((u, v)\) where \(u \in A, v \in B\).

\[ \text{What is the probability of failure in the first step in this example?} \]

A. \(3/20\) 
B. \(3/23\) 
C. \(3/43\) 
D. \(1\)

\[ \text{Pr[fail in round 1]} = \text{Pr[select } u \in A, v \in B] = \frac{k}{|E|} \]

\[ \text{To upper bound failure probability, need lower bound on } |E| \]

Suppose \(G\) is a graph with global minimum cut of size \(k\). Then the degree of every node is always

- A. At least \(k\)
- B. Exactly \(k\)
- C. At most \(k\)
- D. At least \(2m/n\)

\[ \text{Every node has degree } \geq k \implies |E| \geq \frac{1}{2}kn. \]

\[ \text{Pr[fail in round 1]} = \text{Pr[select } u \in A, v \in B] = \frac{k}{|E|} \]

\[ \text{To upper bound failure probability, need lower bound on } |E| \]

- A. At least \(k\)
- B. Exactly \(k\)
- C. At most \(k\)
- D. At least \(2m/n\)

\[ \text{Every node has degree } \geq k \implies |E| \geq \frac{1}{2}kn. \]

\[ \text{Pr[fail in round 1]} = \frac{k}{2kn} = \frac{2}{n} \]

\[ \text{Pr[fail in round 1]} \leq \frac{2}{n} \]

\[ p_{\text{succ}}(1) = \text{Pr[success in round 1]} \geq 1 - \frac{2}{n} = \frac{n-2}{n} \]

\[ \text{Each time, } |E'| \geq k'n'/2, \text{ so success probability } \geq \frac{n'-2}{n'} \]

\[ \text{Success only if success in each round (product of probabilities)} \]

\[ p_{\text{succ}} = p_{\text{succ}}(1)p_{\text{succ}}(2) \cdots p_{\text{succ}}(n-2) \geq \frac{n-2}{n} \cdots \frac{n-3}{n-1} \cdot \frac{1}{n} = \frac{2}{n(n-1)} \]
Contraction Algorithm: Repeat for Success

Theorem. Alg finds global min cut with probability at least \(1/\binom{n}{2}\).

Corollary. If run \(\binom{n}{2}\ln n\) times, success probability is \(\geq 1 - \frac{1}{n}\).

Proof.

\[
\Pr[\text{Fail all } t \text{ times}] \leq \left(1 - \frac{1}{\binom{n}{2}}\right)^t
\]

If \(t = c\binom{n}{2}\) this is at most \(e^{-c}\).

(From calculus: \((1 - 1/x)^x \leq 1/e = \lim_{x \to \infty} (1 - 1/x)^x\))

If run \(\binom{n}{2}\ln n\) times, failure probability \(\leq e^{-\ln n} = \frac{1}{n}\).

Median Find

Problem. Given a set of numbers \(S = \{a_1, \ldots, a_n\}\) the median is the number in the middle if the numbers were sorted.

- If \(n\) is odd then \(k\)th smallest element where \(k = (n + 1)/2\).
- If \(n\) is even then \(k\)th smallest element where \(k = n/2\).

Deterministic algorithm?

- Sort numbers, take \(k\)th smallest.
- \(O(n \log n)\).

Divide and Conquer Algorithm

- Choose splitter (or pivot) \(a_i \in S\).
- Form sets \(S^- = \{a_j : a_j < a_i\}\), \(S^+ = \{a_j : a_j > a_i\}\).

If:

- \(|S^-| = k - 1\): \(a_i\) is the target.
- \(|S^-| \geq k\): recurse on \((S^-, k)\).
- \(|S^-| < k - 1\), recurse on \((S^+, k - (|S^-| + 1))\).

Can do deterministic \(O(n)\) algorithm along same idea [Blum, Floyd, Pratt, Rivest, Tarjan '72]

Today: randomized

How many minimum cuts?

A graph may have several minimum cuts.

When computing probability to succeed, we actually proved more!

- We’ve shown the probability to return
- \(\frac{1}{n(n-1)}\)

\[\binom{n}{2}\ln n\times, \text{success probability is} \geq \frac{2}{n(n-1)}\]

But any two cuts are distinct! Let their number be \(c\)

\[p_{\text{min-cut}} = p_{\text{min-cut1}} + \ldots + p_{\text{min-cutc}} \geq \frac{2}{n(n-1)}\]

But \(p_{\text{min-cut}} \leq 1 \Rightarrow c \leq n(n-1)/2\)

More Generally: \(k\)th Smallest

Problem. Given a set of numbers \(S = \{a_1, \ldots, a_n\}\) and number \(k\), return \(k\)th smallest number. (Assume no duplicates)

Special cases:

- \(k = 1\): minimum element \(O(n)\)
- \(k = n\): maximum element \(O(n)\).

Why is it \(O(n \log n)\) for \(k = n/2\)?

Pseudocode

\[
\text{SELECT}(S,k): \\
\quad \text{Choose splitter } a_i \in S. \\
\quad \text{for each } a_j \in S \text{ do} \\
\quad \quad \text{Put } a_j \in S^- \text{ if } a_j < a_i. \\
\quad \quad \text{Put } a_j \in S^+ \text{ if } a_j > a_i. \\
\quad \quad \text{If } |S^-| = k - 1, \text{ then return } a_i. \\
\quad \quad \text{If } |S^-| \geq k, \text{ return } \text{SELECT}(S^-, k). \\
\quad \quad \text{Else, return } \text{SELECT}(S^+, k - (|S^-| + 1)). \\
\]

Looks kind of like quicksort...

Fact. Algorithm is correct.
How to choose splitter?

We want recursive calls to work on much smaller sets.

▶ Best case, splitter is the median:

\[ T(n) \leq T(n/2) + cn \Rightarrow O(n) \text{ runtime} \]

▶ Worst case, splitter is largest element:

\[ T(n) \leq T(n - 1) + cn \Rightarrow O(n^2) \text{ runtime} \]

▶ Middle case, splitter separates \( \epsilon n \) elements

\[ T(n) \leq T((1 - \epsilon)n) + cn \]

\[ T(n) \leq cn \left[ 1 + (1 - \epsilon) + (1 - \epsilon)^2 + \ldots \right] \leq \frac{cn}{\epsilon} \]

How can we stay close to the best case?

Randomized Splitters

Idea. Choose splitter uniformly at random.

Analysis. Phase \( j \) when \( n(3/4)^{j+1} \leq |S| \leq n(3/4)^j \).

▶ Claim. Expect to stay in phase \( j \) for two rounds.

▶ Call splitter central if separates \( 1/4 \) fraction of elements.

\[ \Pr[\text{central splitter}] = \frac{1}{2} \]

▶ If \( X \) is number of attempts until central splitter,

\[ \mathbb{E}[X] = \sum_{j=1}^{\infty} j \Pr[X = j] = \sum_{j=1}^{\infty} j p(1 - p)^{j-1} \]

\[ = \frac{p}{1 - p} \sum_{j=1}^{\infty} j(1 - p)^j = \frac{p}{1 - p} \frac{1 - p}{p^2} \]

\[ = \frac{1}{p} \]

Applications

▶ Randomized median find in expected linear time

Quicksort (Sketch)

▶ Choose pivot at random. Form \( S^-, S^+ \).

▶ Recursively sort both.

▶ Concatenate together.

Theorem. Quicksort has expected \( O(n \log n) \) time.

Review: Randomized Algorithms

▶ Efficient in expectation

▶ Optimal with high probability

▶ Break (undesired) symmetry

▶ Show some solution exists, or derive bound on number

Types of randomized algorithms:

▶ Fail with some small probability (Monte Carlo)

▶ Always succeed, but running time may be large (Las Vegas)