AADS, Lecture 4

Randomized Algorithms

Jacob Holm (`jaho@di.ku.dk`)

November 27th 2024

Good afternoon. My name is Jacob Holm.

You can help by asking questions during class if there is anything that is not clear.

Remember, if it is not clear to you, then it is probably also unclear to at least one other person in the room.

You can help more than just yourself by asking for clarification.

I am also teaching the Randomized Algorithms course, and the next two lectures are a tiny taste of that.

Why Randomized Algorithms?

- Faster, but weaker guarantees.
- Simpler code, but harder to analyze.
- Sometimes only option, e.g. Big Data, Streaming, Machine Learning, Security, (Differential) Privacy, etc.

Therefore this course!

Todays Lecture

- Quicksort
  - Linearity of expectation
  - Expectation of indicator variable
- Min-Cut
  - Conditional probabilities
  - Time/error probability tradeoff
- Las Vegas vs Monte Carlo
- Summary

AADS Lecture 4 (RA), Part 1

Quicksort

Basic Quicksort [Hoare]

function $\mathrm{QS}(S = \{s_1, \ldots, s_n\})$
 ▷ Assumes all elements in S are distinct.
  if $|S| \leq 1$ **then**
   return list(S)
  else
   Pick pivot $x \in S$, (How?)
   $L \leftarrow \{y \in S \mid y < x\}$
   $R \leftarrow \{y \in S \mid y > x\}$
   return $\mathrm{QS}(L)+[x]+\mathrm{QS}(R)$

For each $y \in S \setminus \{x\}$, compare to y to x once

Lemma

For any pivoting strategy, QS *correctly sorts the numbers.*

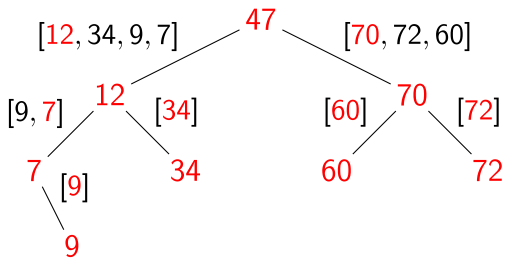
Proof.

By induction on n. $n = 0, 1$ is trivial, so assume it holds for up to $n - 1$ numbers. Then by our induction hypothesis $\mathrm{QS}(L)$ and $\mathrm{QS}(R)$ are sorted, so $\mathrm{QS}(L)+[x]+\mathrm{QS}(R)$ is sorted. □

Q: Does anyone see what essential part is missing from this description?

Quicksort Example 1

Sorting $S = [70, 12, 34, 47, 9, 72, 60, 7]$.



Total #comparisons: $4 + 3 + 2 + 1 + 1 + 1 + 1 = 13$

Assuming we always pick the "middle" element we have $T(n) \leq 2T(n/2) + \mathcal{O}(n)$, so by the Master Theorem $T(n) \in \mathcal{O}(n \log n)$.

As you can see this looks rather like a balanced binary search tree, so you might expect the number of comparisons to be small. Something like $n \log n$.

Quicksort Example 2

Sorting $S = [70, 12, 34, 47, 9, 72, 60, 7]$.

7 $[70, 12, 34, 47, 9, 72, 60]$

9 $[70, 12, 34, 47, 72, 60]$

12 $[70, 34, 47, 72, 60]$

34 $[70, 47, 72, 60]$

47 $[70, 72, 60]$

60 $[70, 72]$

70 $[72]$

72

Total #comparisons: $7 + 6 + 5 + 4 + 3 + 2 + 1 = 28$

Assuming we always pick an "extreme" element we have $T(n) = T(0) + T(n-1) + \Theta(n)$, and thus $T(n) \in \Theta(n^2)$.

As you can see this looks like an extremely unbalanced binary search tree, so you might expect the number of comparisons to be large. Something like n^2.

Randomized Quicksort

function $\textsc{RandQS}(S = \{s_1, \ldots, s_n\})$
▷ Assumes all elements in S are distinct.
  if $|S| \leq 1$ **then**
    return S
  else
    Pick pivot $x \in S$, uniformly at random
    $L \leftarrow \{y \in S \mid y < x\}$
    $R \leftarrow \{y \in S \mid y > x\}$
    return $\textsc{RandQS}(L) + [x] + \textsc{RandQS}(R)$

For each $y \in S \setminus \{x\}$, compare to y to x once

Randomized Quicksort, Analysis

Q: What is the expected number of comparisons?

Theorem

$\mathbb{E}[\#\textit{comparisons}] \in \mathcal{O}(n \log n)$

Let $[S_{(1)}, \ldots, S_{(n)}] := \textsc{RandQS}(S)$.
For $i < j$ let X_{ij} be the number of times that $S_{(i)}$ and $S_{(j)}$ are compared. We can then compute

$$\#\text{comparisons} = \sum_{i<j} X_{ij}$$

$$\mathbb{E}[\#\text{comparisons}] = \mathbb{E}\Big[\sum_{i<j} X_{ij}\Big] = \sum_{i<j} \mathbb{E}[X_{ij}]$$

Uses *linearity of expectation*:

$$\mathbb{E}[A+B] = \mathbb{E}[A] + \mathbb{E}[B]$$

Important: $S_{(i)}$ is the ith element in the final sorted order.
Note that the $\sum_{i<j}$ is really a shorthand for $\sum_{1\leq i<j\leq n}$, or even more explicit $\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}$.

Randomized Quicksort, Analysis

Observe that $X_{ij} \in \{0,1\}$ (why?).

Since $X_{ij} \in \{0,1\}$, it is an *indicator variable* for the event that $S_{(i)}$ and $S_{(j)}$ are compared. Let p_{ij} be the probability of this event. Then

$$\begin{aligned}\mathbb{E}[X_{ij}] &= \sum_{x\in\{0,1\}} \Pr[X_{ij} = x] \cdot x \qquad\qquad (\text{Def. of } \mathbb{E})\\ &= (1 - p_{ij}) \cdot 0 + p_{ij} \cdot 1 = p_{ij}\end{aligned}$$

Thus *the expectation of an indicator variable equals the probability of the indicated event.*

Therefore

$$\mathbb{E}[\#\text{comparisons}] = \sum_{i<j} \mathbb{E}[X_{ij}] = \sum_{i<j} p_{ij}$$

Randomized Quicksort, Analysis

Lemma

$S_{(i)}$ and $S_{(j)}$ are compared iff $S_{(i)}$ or $S_{(j)}$ is first of $S_{(i)}, \ldots, S_{(j)}$ to be chosen as pivot.

Proof.

Each recursive call returns some sublist $[S_{(a)}, \ldots, S_{(b)}]$. Let $x = S_{(c)}$ be the pivot.

Suppose $a \leq i < j \leq b$. $\boxed{a \,|\, \cdots \,|\, i \,|\, \cdots \,|\, j \,|\, \cdots \,|\, b}$

$c < i$ or $c > j$: $S_{(i)}$ and $S_{(j)}$ not compared now, but together in recursion. Recursion stops when $i \leq c \leq j$.

$i < c < j$: $S_{(i)}$ and $S_{(j)}$ never compared.

$c \in \{i, j\}$: $S_{(i)}$ and $S_{(j)}$ compared once. $\square$

Randomized Quicksort, Analysis

Thus, p_{ij} is the conditional probability of picking $S_{(i)}$ or $S_{(j)}$ given that the pivot is picked uniformly at random in $\{S_{(i)}, S_{(i+1)}, \ldots, S_{(j)}\}$:

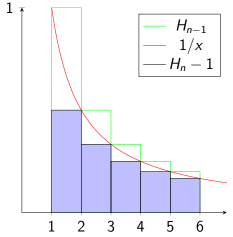
$$\begin{aligned} p_{ij} &= \Pr[c \in \{i, j\} \mid c \in \{i, i+1, \ldots, j\} \text{ u.a.r.}] \\ &= \frac{2}{|\{i, i+1, \ldots, j\}|} = \frac{2}{j+1-i} \end{aligned}$$

It follows that

$$\mathbb{E}[\#\text{comparisons}] = \sum_{i<j} p_{ij} = \sum_{i<j} \frac{2}{j+1-i}$$

Randomized Quicksort, Analysis

$$
\begin{aligned}
\mathbb{E}[\#\text{comparisons}] &= \sum_{i<j} \frac{2}{j+1-i} \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j+1-i} \\
&= \sum_{i=1}^{n-1} \sum_{k=2}^{n+1-i} \frac{2}{k} < \sum_{i=1}^{n} \sum_{k=2}^{n} \frac{2}{k} \\
&= 2n \sum_{k=2}^{n} \frac{1}{k} = 2n\Big(\Big(\sum_{k=1}^{n} \frac{1}{k}\Big) - 1\Big) = 2n(H_n - 1) \\
&\leq 2n \int_1^n \tfrac{1}{x} dx = 2n \ln n \in \mathcal{O}(n \log n)
\end{aligned}
$$



$H_n - 1 \leq \int_1^n \frac{1}{x} dx = \ln n \leq H_{n-1}$

- From before.
- Expanding the $\sum_{i<j}$ notation.
- The denominator $k = j + 1 - i$ takes each value from $2, \ldots, n + 1 - i$ once.
- Since all terms are positive, adding more terms can only increase the value
- Moving 2 outside the sums and noting that the inner sum does not depend on i.
- Adding and subtracting the term for $k = 1$.
- Using the definition of H_n.
- Observing that $H_n - 1 \leq \int_1^n \frac{1}{x} dx = \ln n \leq H_{n-1} = H_n - \frac{1}{n}$.

Randomized Quicksort, Summary

When $|S| = n$, the expected number of comparisons done by $\textsc{RandQS}(S)$ is less than $2nH_n \in \mathcal{O}(n \log n)$ *for any input*.

Even stronger (see Problem 4.14), we can show that the number of comparisons is $\mathcal{O}(n \log n)$ *with high probability*.

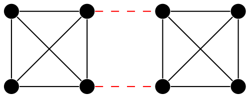
AADS Lecture 4 (RA), Part 2

Min-Cut

Min-Cut

This is a related, but different notion of Min-Cut than in the "Max-Flow Min-Cut theorem" from lecture 1 and 2.

Problem: Given a connected graph $G = (V, E)$



Find smallest $C \subseteq E$ that splits G.

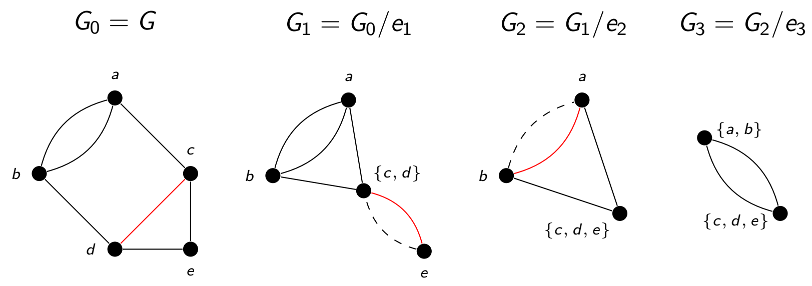
C is called a *(global) min-cut*, and $\lambda(G) := |C|$ is the *edge connectivity* of G.

Randomized Min-Cut [Karger & Stein]

function $\textsc{RandMinCut}(V, E)$
  while $|V| > 2$ and $E \neq \emptyset$ **do**
    Pick $e \in E$ uniformly at random.
    Contract e and remove self-loops.
  return E

Randomized Min-Cut, Example

function $\textsc{RandMinCut}(V, E)$
  while $|V| > 2$ and $E \neq \emptyset$ **do**
    Pick $e \in E$ uniformly at random.
    Contract e and remove self-loops.
  return E



Randomized Min-Cut, Analysis

function RANDMINCUT(V, E)
  while $|V| > 2$ and $E \neq \emptyset$ **do**
    Pick $e \in E$ uniformly at random.
    Contract e and remove self-loops.
  return E

Lemma

RANDMINCUT(G) *always returns a cut.*

Proof.

Proof by induction on the number k of iterations of the loop (note $k \leq n - 2$). If $k = 0$ it is trivial, so suppose that it is true for up to $k - 1$ iterations. The first iteration constructs graph G' by contracting an edge from G and removing self-loops, and then do at most $k - 1$ futher iterations starting from G' so by the induction hypothesis we return a cut in G'. But every such cut is also a cut in G. $\square$

Randomized Min-Cut, Analysis

```
function RandMinCut(V, E)
    while |V| > 2 and E ≠ ∅ do
        Pick e ∈ E uniformly at random.
        Contract e and remove self-loops.
    return E
```

Observation

$\textsc{RandMinCut}(G)$ *may return a cut of size* $> \lambda(G)$.

Lemma

A specific min-cut C *is returned iff no edge from* C *was contracted.*

Randomized Min-Cut, Analysis

Theorem

For any min-cut C, the probability that $\textsc{RandMinCut}(G)$ *returns C is* $\geq \frac{2}{n(n-1)}$.

Let $e_1, \ldots, e_{n-2}$ be the contracted edges, let $G_0 = G$ and $G_i = G_{i-1}/e_i$.
Let $\mathcal{E}_i$ be the (good) event that $e_i \notin C$.
C is returned iff $\mathcal{E}_1 \cap \cdots \cap \mathcal{E}_{n-2}$.

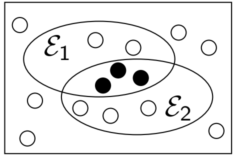
Goal: $\Pr[\mathcal{E}_1 \cap \cdots \cap \mathcal{E}_{n-2}] \geq \frac{2}{n(n-1)}$

- In words, $\mathcal{E}_i$ is the event that the ith edge contracted is not in C, i.e., the ith contraction does not destroy C.
- $\mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{n-2}$ is thus the event that C is not destroyed in any step of the algorithm.

Conditional Probabilities

Given events $\mathcal{E}_1, \mathcal{E}_2$ with $\Pr[\mathcal{E}_1] > 0$, the *conditional probability* of $\mathcal{E}_2$ given $\mathcal{E}_1$ is defined as

$$\Pr[\mathcal{E}_2|\mathcal{E}_1] = \frac{\Pr[\mathcal{E}_1 \cap \mathcal{E}_2]}{\Pr[\mathcal{E}_1]}$$



It follows that

$$\Pr[\mathcal{E}_1 \cap \mathcal{E}_2] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2|\mathcal{E}_1]$$

And in general for events $\mathcal{E}_1, \ldots, \mathcal{E}_k$

$$\Pr[\cap_{i=1}^{k} \mathcal{E}_i] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2|\mathcal{E}_1] \cdots \Pr[\mathcal{E}_k| \cap_{i=1}^{k-1} \mathcal{E}_i]$$

This is easy to prove by induction.

Randomized Min-Cut, Proof

$$\begin{aligned}
&\Pr[\text{specific min-cut } C \text{ returned}] \\
&\quad = \Pr[\mathcal{E}_1 \cap \cdots \cap \mathcal{E}_{n-2}] \\
&\quad = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 | \mathcal{E}_1] \cdots \Pr[\mathcal{E}_{n-2} | \mathcal{E}_1 \cap \cdots \cap \mathcal{E}_{n-3}] \\
&\quad = \prod_{i=1}^{n-2} p_i \quad \text{where } p_i = \Pr[\mathcal{E}_i | \mathcal{E}_1 \cap \cdots \cap \mathcal{E}_{i-1}]
\end{aligned}$$

Randomized Min-Cut, Proof

$G_i = (V_i, E_i)$ has $n_i = n - i$ vertices. (why?)
Contractions can not decrease the min-cut size (why?)
so $\lambda(G_i) \geq |C|$.
It follows that each vertex v of G_i has degree $d_i(v)$ at least $|C|$. (why?)
Summing up all degrees of G_i,

$$|E_i| = \frac{1}{2} \sum_{v \in V_i} d_i(v) \geq \frac{1}{2} n_i |C|.$$

- Each contraction reduces the number of vertices by 1.
- Every cut in G_i is a cut in G_{i-1} and therefore in G.
- Note that the edges incident to a vertex v form a cut and so $d_i(v) \geq \lambda(G_i) \geq |C|$.
- We use that each edge is counted twice in the sum $\sum_{v \in V_i} d_i(v)$.

Randomized Min-Cut, Proof

We have shown that $G_i = (V_i, E_i)$ has $n_i = n - i$ vertices and at least $|E_i| \geq \frac{1}{2} n_i |C|$ edges. We want to lower bound

$$p_i = \Pr[\text{uniformly random } e \in E_{i-1} \text{ is not in } C \mid \cap_{j=1}^{i-1} \mathcal{E}_j]$$

The complementary probability, i.e. the probability of picking an edge of C in the ith iteration, given that no edge of C has been picked in a previous iteration, is

$$\begin{aligned} 1 - p_i &= \Pr[\text{uniformly random } e \in E_{i-1} \text{ is in } C \mid \cap_{j=1}^{i-1} \mathcal{E}_j] \\ &= \frac{|C|}{|E_{i-1}|} \leq \frac{|C|}{\frac{1}{2} n_{i-1} |C|} = \frac{2}{n_{i-1}} = \frac{2}{n - (i-1)} \\ \Rightarrow p_i &\geq 1 - \frac{2}{n-i+1} = \frac{n-i-1}{n-i+1} \end{aligned}$$

We have shown upper bound

$$1 - p_i \leq \frac{2}{n-i+1}$$

so now we can lower bound p_i.

Randomized Min-Cut, Proof

Telescoping product.

$$
\begin{aligned}
&\Pr[C \text{ returned}] \\
&\quad = \prod_{i=1}^{n-2} p_i \quad \text{where } p_i = \Pr[\mathcal{E}_i | \mathcal{E}_1 \cap \cdots \cap \mathcal{E}_{i-1}] \\
&\quad \geq \prod_{i=1}^{n-2} \frac{n-1-i}{n+1-i} \\
&\quad = \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdots \frac{3}{5} \cdot \frac{2}{4} \cdot \frac{1}{3} \\
&\quad = \frac{2}{n(n-1)}
\end{aligned}
$$

Randomized Min-Cut, Summary

So for given min-cut C, $\Pr[C \text{ is returned}] \geq \frac{2}{n(n-1)}$.

Is this tight? I.e. do we have examples matching this bound? Yes! Consider the cycle C_n on n vertices. Every one of the $\binom{n}{2} = \frac{n(n-1)}{2}$ pairs of edges is a min-cut and all pairs are equally likely to be returned.

Is this probability good?

How can we improve it?

Randomized Min-Cut, Tradeoff

Imagine calling $\textsc{RandMinCut}(G)$ $t\frac{n(n-1)}{2}$ times and letting $C^\star$ be the smallest cut returned.

$$\Pr[C^\star \text{ is not a min-cut}] \le \left(1 - \frac{2}{n(n-1)}\right)^{t\frac{n(n-1)}{2}}$$

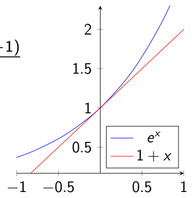
$$\le \left(e^{-\frac{2}{n(n-1)}}\right)^{t\frac{n(n-1)}{2}}$$

(This uses that $1 + x \le e^x$ for all $x \in \mathbb{R}$, see Proposition B.3.1)

$$= e^{-t}$$

$$\Pr[C^\star \text{ is a min-cut}] \ge 1 - e^{-t}$$

$$= 1 - n^{-c} \qquad (\text{If we set } t = c \ln n)$$



Thus for any $c > 0$ if we repeat $c \cdot \frac{n(n-1)}{2} \cdot \ln n$ times, the probability of getting a min-cut is at least $1 - n^{-c}$. We call this *high probability of success*.

- In each call to $\textsc{RandMinCut}(G)$, the probability that a min-cut is not returned is at most $1 - \frac{2}{n(n-1)}$.
- Since the calls to $\textsc{RandMinCut}(G)$ are independent, the probability that no min-cut is among the cuts returned is the product.
- $1 + x \le e^x$
- Choosing e.g. $t = 21$ we reduce the error probability to around one in a billion.
- Choosing $t = c \ln n$ for constant c, we get a *high probability of success*, namely at least $1 - e^{-c\ln n} = 1 - 1/n^c$.
- We thus get a tradeoff between running time and probability of success.

Randomized Min-Cut, Simple implementation

In practice, using a "Union-Find" data structure.

```
function RandMinCut(V, E)
    for u ∈ V do
        make-set(u)
    C ← ∅, π ← a random permutation of E, r ← |V|
    for uv ∈ E in the order π do
        p_u ← find(u), p_v ← find(v)
        if p_u ≠ p_v then
            if r > 2 then
                r ← r − 1
                union(p_u, p_v)
            else
                C ← C ∪ {uv}
    return C
```

The running time for this is $\mathcal{O}(m\alpha(n))$. Running it $\mathcal{O}(n^2 \log n)$ times to get high probability takes $\mathcal{O}(n^2 m\alpha(n) \log n)$ time.

Deterministic Min-Cut

Pick arbitrary $s \in V$. For each $t \in V \setminus \{s\}$ compute max-flow from s to t. Return the minimum.

What is the running time? We run Ford-Fulkerson $n-1$ times. Each run takes $\mathcal{O}(m|f^*|)$ time where $|f^*| \leq m$. In total $\mathcal{O}(nm^2)$ time. For dense graphs this is much worse than $\mathcal{O}(n^2 m\alpha(n)\log n)$.

Best algorithm known (2024): $\widetilde{\mathcal{O}}(m)$-time algorithm by Kawarabayashi and Thorup [JACM 2019].

Slightly improved by Henzinger, Rao and Wang [SICOMP20].

Extended to weighted graphs by Henzinger, Li, Rao and Wang [SODA 2024]

Edmonds-Karp does not help us here, at least not without a better analysis. For unit-capacity graphs like this one, the bound $\mathcal{O}(m|f^*|) = O(m^2)$ is better than the $\mathcal{O}(nm^2)$ we get for Edmonds-Karp. On the other hand, since Edmonds-Karp *is* a version of Ford-Fulkerson, you can/should still use it, as the same improved analysis applies.

Las Vegas vs Monte Carlo

What is the main difference between the guarantees of RandQS and RandMinCut?

Las Vegas: Always returns correct answer. #steps used is a random variable.

Monte Carlo: Some probability of error. #steps used may be random or not.

Converting L.V. $\leftrightarrow$ M.C.

As part of Assignment 2 you will prove that we can sometimes convert a Monte Carlo algorithm into a Las Vegas algorithm.

How about the other direction? Can we always take a Las Vegas algorithm running in expected $\mathcal{O}(f(n))$ time and turn it into a Monte Carlo algorithm running in worst case $\mathcal{O}(f(n))$ time?

Summary

- We analyzed a Las Vegas randomized algorithm (RandQS).
- We analyzed a Monte Carlo randomized algorithm (RandMinCut).
- We discussed how the two types of algorithms are related and may often be converted into each other.

Next time

Hashing.