CSEP 521: Applied Algorithms
Lecture 2 Randomized Algorithms

Richard Anderson

January 7, 2021

Announcements

- Contact info:
  - Richard Anderson, anderson@cs.washington.edu
  - Oscar Sprumont, osprum@cs.washington.edu
- Office hours (Zoom)
  - Richard Anderson, Monday 11 am – noon, Friday 2 pm – 3 pm
  - Oscar Sprumont, Wednesday 11 am – noon, Friday 11 am – noon
- Edstem Discussion board set up
  - https://edstem.org/us/courses/3570/discussion/
- Course readings
  - Starting to put of references and links on website (under readings)

Randomized Algorithms

- Randomization comes up in many different ways in computation
- Illustrate different ways randomization is used by example algorithms
  - Algorithmic Techniques + Analysis Techniques
  - Demonstrate some incredibly cool idea that are foundational for computing
- K-th largest problem (median problem)
  - Randomization to avoid bad case
  - Relation between average case and randomized analysis
- Random sampling algorithms (again for the median)

Probability theory

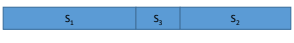
- Basic ideas:
  - Sample space and events
  - Random variables
  - Expectation: E(X) – average value of X
  - Linearity of expectation: E(X + Y) = E(X) + E(Y)
  - Conditional Probability: P(A | B)
  - Independence: P(A | B) = P(A)
- Great review videos
  - Tim Roughgarden (ex-Stanford), Videos 7.1 and 7.2
- Many other references

Finding the k-th largest

- Given n numbers and an integer k, find the k-th largest
- If k = n/2 this is computing the median
- Obviously, we can solve this problem by sorting (in time O(n log n)) but can we do better
- Quicksort idea, but only one recursive call
- This will still have the same pathological case

QSelect(A, k)

```
QSelect(A, k){
        Choose element x from A
        S1 = {y in A | y < x}
        S2 = {y in A | y > x}
        S3 = {y in A | y = x}
        if (|S2| >= k)
                return QSelect(S2, k)
        else if (|S2| + |S3| >= k)
                return x
        else
                return QSelect(S1, k - |S2| - |S3|)
}
```



Quick Select

- Same worst case as Quick Sort
- Random pivot will give an O(n) solution (Expected time)
- Deterministic solution (due to BFPRT) not practical
- Exact evaluation of Quick Select recurrence is very difficult

Evaluating Quick Select

- Deterministic recurrence: T(1) = 1; T(N) = T(aN) + bN for a < 1
- What is the chance that for a random pivot, the subproblem is of size at most aN (for an appropriate a < 1)

Analysis

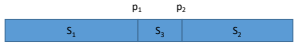
- A random pivot reduces the problem to less than ¾ the size with probability at least ½
- Recurrence T(n) <= ½ T(3n/4) + ½ T(n) + cn
- Recurrence for expected number of steps
- This recurrence gives an upper bound on the number of steps for randomized selection

Selection results

- O(n) expected time median algorithm
- Practical algorithm
- How many comparison to find the median?

Sampling algorithm for finding median

- Two pivot algorithm
  - Choose pivot p_1, such that p_1 is less than the median
  - Choose pivot p_2, such that p_2 is greater than the median
- Identify the elements between p_1 and p_2



- Find the k-th largest element of S_3 where k = n/2 - $|S_3|$

Finding the pivots

- Select a set A of m random values from the input
- Let p_1 be the element of rank m/2 + r in A
- Let p_2 be the element of rank m/2 – r in A
- Claim: if m << n and $|S_3|$ << n then the comparisons are (3/2)n + o(n)
- Choose m = $n^{2/3}$ and r = $n^{1/3}$

Breakout session

- Suppose P is a random permutation (on 1..n), what is the expected number of elements k such that P[k] = k
- This could be given with a story line, such as saying that when everyone in an office goes back to work, they are assigned a random desk, what is the number of people that are assigned to their old desk
- Discuss as a group, and if you come up with a solution, make sure all group members understand the solution

Randomized Algorithms

- How can random choices be used to make algorithms efficient?
- Suppose you have a minimization problem, where given a solution you can easily compute the value of the solution
- Now suppose you have an algorithm, which returns a specific optimal solutions with probability p.
- Can you find an optimal solution? What can go wrong?

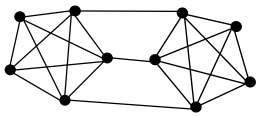
Mincut problem: Karger's Algorithm

- Graph problem
- Random result – might be correct
- Given an undirected graph, partition the vertices into two non-empty pieces to minimize the edges between the pieces
- Undirected graph with parallel edges

Notes:
Different from the standard mincut problem presented along with Network Flow
The algorithm presented is not very efficient, but can be improved to be faster than any known deterministic algorithm for the problem

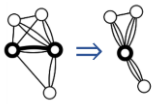
Minimum Cut Problem

- Given an undirected graph with parallel edges, find non-empty sets S and S' that minimized the number of edges between S and S'



Edge contraction operation

Contraction operation:
For an edge $e \in E$, write G/e for the new graph formed by contracting the edge e.



Observation: The number of edges in the mincut of G/E is at least as great as the number of edges in a mincut of G

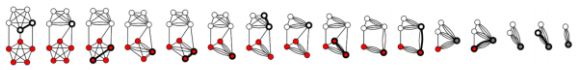
Randomized Algorithm

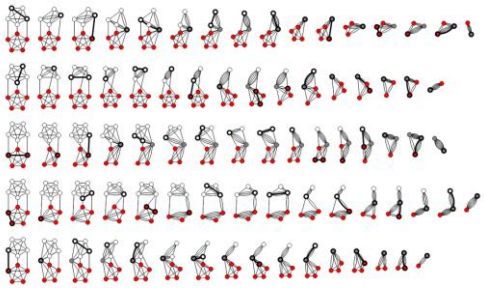
```
procedure contract(G = (V, E)):
while |V| > 2
    choose e ∈ E uniformly at random
    G ← G/e
return the only cut in G
```

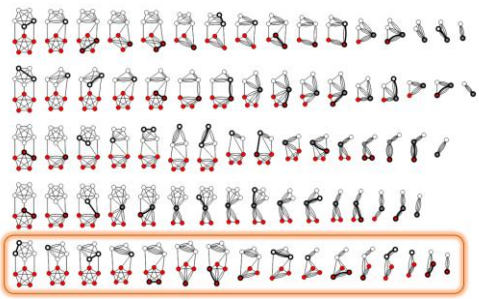
How many times does the while loop execute?

$|V| - 2$ times

```
procedure contract(G = (V, E)):
while |V| > 2
    choose e ∈ E uniformly at random
    G ← G/e
return the only cut in G
```







Analysis

- Let C be a min cut (where C is the set of edges between sets of vertices S and S' and |C| = k). What is the probability that C survives the contraction process?
- G has at least kn/2 edges, otherwise G would have a vertex of degree less than k (and consequently a cut of size less than k).
- Probability that the first edge picked is in C is at most k / (nk/2) = 2/n

Conditional Probabilities

E_i: Event of not picking an edge from C on the i-th step

Probability that no edges from C are picked:

$p = \text{Prob}[E_1]\ \text{Prob}[E_2 \mid E_1]\ \text{Prob}[E_3 \mid E_1 E_2] \ldots \text{Prob}[E_{n-2} \mid E_1 E_2 \ldots E_{n-3}]$

$\text{Prob}[E_i \mid E_1 E_2 \ldots E_{i-1}] >= 1 - 2 / (n - i + 1) = (n - i - 1) / (n - i + 1)$

$p >= (n-2)/ n * (n-3) / (n-1) * (n-4) / (n-2) * \ldots * 2/4 * 1/3 = 2 / n(n-1)$

Min Cut Result

- Probability that C survives is at least 2 / n(n-1) on one iteration
- If we repeat the algorithm n(n-1) / 2 times we have probability at least 1 - 1/e that C has survived at least once
- Mathematical fact: $(1 - 1/k)^k < 1/e$
- If we run $O(n^2 \log n)$ iterations, we have the probability of at least $1 - 1/n$ of finding a mincut