1 Karger-Stein

Recall from lecture Karger’s algorithm to find the minimum cut in an unweighted graph $G$:

procedure CONTRACT($G = (V, E)$)
    if $|V| = 2$ then output the only cut
    else
        pick a random edge $e$
        contract the edge $e$ to form $G' = (V', E')$
    return CONTRACT($G'$)
end

We showed in lecture that, for any minimum cut $C$, this randomized algorithm outputs $C$ with probability $1/(\binom{n}{2})$ and moreover the contraction can be implemented to run in $O(n^2)$ time.

We now turn to Karger and Stein’s variant which outputs a minimum cut with higher probability. Notice that in the first round of contraction the probability that some mincut $C$ survives is pretty good: at least $1 - 2/n$. Meanwhile toward the end, the probability $C$ survives starts getting much farther away from one: we can only say it is at least $1/3$ in the last iteration.

The idea of Karger and Stein was to run contraction not until we get down to two vertices, but rather until we get down to $t$ vertices. Then the probability that $C$ has still survived up until this point, by contraction, is at least $t(t-1)/(n(n-1))$. By picking $t = \lceil n/\sqrt{2} \rceil + 1$, we have that the probability that $C$ survives after contracting down to $t$ vertices is at least $1/2$. The Karger-Stein algorithm, henceforth known as KS, is then as follows:
procedure KS\( (G = (V, E)) \)
if \(|V| < 6\) then try all \(2^{|V| - 1}\) cuts and return the smallest one
else
\( t \leftarrow \lceil n/\sqrt{2} \rceil + 1 \)
Obtain \(G_1\) by contracting \(G\) \(n - t\) times to obtain \(t\) vertices
Obtain \(G_2\) by contracting \(G\) \(n - t\) times to obtain \(t\) vertices
(using independent randomness from last step)
\( C_1 \leftarrow KS(G_1) \)
\( C_2 \leftarrow KS(G_2) \)
return the smaller of the two cuts \(C_1\) and \(C_2\)
end

The reason we make \(|V| < 6\) a special case is that \(\lceil n/\sqrt{2} \rceil + 1 \geq n\) for \(n < 6\), and thus the recursive calls in the else statement would not actually shrink the problem size for \(n < 6\).

**Exercise 1.** What is the running time of the above algorithm?

The key benefit though will be that KS has much better success probability. Whereas the vanilla contraction algorithm had success probability at least \(1/(n^2)\), KS has success probability at least \(\Omega(1/\log n)\).

**Exercise 2.** Show that KS outputs a minimum cut with probability \(\Omega(1/\log n)\).

Given the above lemma, as with the analysis for the vanilla contraction algorithm, we see that it suffices to run KS \(\Theta(\log n \log(1/P))\) times to have success probability \(1 - P\). Since the KS running time itself is \(O(n^2 \log n)\), that implies that the total running time to obtain success probability \(1 - P\) is \(O(n^2 (\log n)^2 \log(1/P))\).
2 Counting Bloom Filters

Recall from last section the **bloom filter**, a data structure for membership problems that are more space-efficient than hashing schemes (since we store just 1 bit per slot in the hash table), but at the cost of some probability of false positives. As review, Bloom filters involve $k$ hash functions $f_1, \cdots, f_k$ mapping into $m$ bits, where bits $f_1(x), \cdots, f_k(x)$ are set to 1 when adding $x$ to the set, and similarly $f_1(x), \cdots, f_k(x)$ are checked to be 1 when looking up $x$.

We showed last section that these simple Bloom filters could not support deletions, since deletions could lead to false negatives. A slightly modified data structure, called the counting bloom filter, uses more space but supports deletions. Now, instead of each of the $m$ slots having just 1 bit, they each have $j$ bits, and adding $x$ to the set increments the counters at slots $f_1(x), \cdots, f_k(x)$, and deleting $x$ from the set decrements the counters at those slots, after checking for $x$’s membership. It’s possible, however, for these counters to overflow:

**Exercise 3.** What is the probability a counting bloom filter with $k$ hash functions mapping into $m$ $j$-bit counters overflows at a specific counter after adding $n$ elements?

3 Random Walks

A random walk is an iterative process on a set of vertices $V$. In each step, you move from the current vertex $v_0$ to each $v \in V$ with some probability. The simplest version of a random walk is a one-dimensional random walk in which the vertices are the integers, you start at 0, and at each step you either move up one (with probability 1/2) or down one (with probability 1/2).

**2-SAT:** In lecture, we gave the following randomized algorithm for solving 2-SAT. Start with some truth assignment, say by setting all the variables to false. Find some clause that is not yet satisfied. Randomly choose one the variables in that clause, say by flipping a coin, and change its value. Continue this process, until either all clauses are satisfied or you get tired of flipping coins.

We used a random walk with a completely reflecting boundary at 0 to model our randomized solution to 2-SAT. Fix some solution $S$ and keep track of the number of variables $k$ consistent with the solution $S$. In each step, we either increase or decrease $k$ by one. Using this model, we showed that the expected running time of our algorithm is $O(n^2)$.

**Exercise 4.** Solve the 2SAT recurrence relations analytically (in lecture we just plugged in a function to see it was a solution):
\[
T(n) = 0
\]
\[
T(i) = \frac{T(i-1)}{2} + \frac{T(i+1)}{2} + 1, \ i \geq 1
\]
\[
T(0) = T(1) + 1.
\]

**Exercise 5.** (Challenging) Now suppose that there are \( n + 1 \) people in a circle numbered 0, 1, \ldots, \( n \). Person 0 starts with a bag of candy. At each step, the person with the bag of candy passes it either left or right with equal probability. The last person to receive the bag wins (and gets to keep all the candy). So if you were playing, then you would want to receive the bag only after everyone else has received the bag at least once. What is the probability that person \( i \) wins?