1 Definitions

1.1 Las Vegas

A randomized algorithm that always gives the correct result, but whose running time distribution is studied (running time varies per run).

It is a monte-carlo algorithm with error probability 0.

1.2 Monte Carlo

A randomized algorithm that might cause incorrect result, but whose probability of being incorrect can be bound.

1.2.1 One side errors

For decision problems, if there is a non-zero probability that it errs on one of the possible outputs(might only error to produce yes).

1.2.2 Two-sided error

For decision problems, there is a non-zero probability that it errs

2 Randomized quicksort

A very simple quicksort algorithm for sorting a sequence S of n number could work as follows

1. Pick element y
2. Compare y to the elements of S such that S is partitioned into two sets S_1 and S_2, where $S_1$are those elements smaller than y and S_2those larger
3. Do this recursively to sort S_1, S_2 and return S_1, y, S_2,

The question is how to pick the element $y$fast. On way is to just pick it randomly.

2.1 Expected running time

Randomized quicksorts works by selecting the pivot element in line 1 uniformly at random. We can analyze the expected running time by looking at the number of comparisons the randomized algorithm will perform. That is, we analyze the time spent at step 2 of the algorithm.

Define S^i to denote the i'th rank of S (the i'th smallest element of S). Additionally the random variable X_{ij} is defined to be 1 of S^i and S^j are compared and 0 otherwise. Elements are only compared with the pivot element y, meaning S^i and $S^j$are only compared if one of them is the pivot element. After comparing S^i to S^j we put one of them in the sublist S_i to be recursively sorted and S^i and $S^j$are thereofore only compared once. Consequently, X_{ij} is a count of

the number of times S^i and S^j are compared. Since each element S^i is at most compared to all other elements S^j once, we get the total number of comparisons is

$$\sum_{i=1}^{n} \sum_{j>i} X_{ij}$$

We can take the expectation of the number of comparisons to get the expected number of comparisons

$$E\left[\sum_{i=1}^{n} \sum_{j>i} X_{ij}\right] = \sum_{i=1}^{n} \sum_{j>i} E\left[X_{ij}\right] = \sum_{i=1}^{n} \sum_{j>i} p_{ij}$$

where the first inequality uses linearity of expectation, the second uses that the expectation of a Bernoulli r.v is the probability of success and $p_{ij}$is the probability of success. All we need to do now is figure out what p_{ij} is.

We analyze the execution of RandQS as a binary search tree T, with each node being the pivot element y selected and the left subtree the elements compared to be less than y and the right subtree greater than y. Note, the elements of the left subtree are never compared to the elements of the right subtree, but only the parent node (pivot element). This confirms that S^i and S^j are only compared if one of them is the pivot. In the tree setting, it means that S^i and S^j are only compared if one of them is the ancestor to the other.

2.2 Lemma

We can view the binary search tree as a random search tree, since we choose each pivot at random, so the nodes of the trees are chosen uniformly at random. The book calls this permutation π, but I find this a bit hard to understand. Instead i use this lemma.

The elements S^i and $S^j$are only compared if and only if S^i or S^j are chosen as pivot element any element in the range $S^i, ..., S^j$ (not the bi-implication).

Proof

Let S^k be chosen as the pivot element in the current iteration of the algorithm, such that $S^i, ..., S^k, ..., S^j$. We have two cases

1. $i < k$ and $k < j$ in which case S^i is put in the left subtree implicitly generated by quicksort and S^j in the right subtree. In either case, since two subtrees are never compared S^i and S^j are never compared.

2. $k \in \{i, j\}$ in which case either S^j or S^i is the pivot the other is compared to and put into the left or right subtree

2.3 Running time cont

Any element in $S^i, ..., S^j$ are equally likely to be picked as the pivot, since the pivot is picked uniformly at random. Hence there are $j-i+1$ possible ways of picking an element. However, picking either S^i or S^j (2 cases) means $X_{ij} = 1$. Hence we get

$$p_{ij} = \frac{2}{j-i+1}$$

Now we can continue with the expectation on number of comparisons

$$\begin{aligned} E\left[\sum_{i=1}^{n}\sum_{j>i} X_{ij}\right] &= \sum_{i=1}^{n}\sum_{j>i} p_{ij} = \sum_{i=1}^{n}\sum_{j>i}\frac{2}{j-i+1} \\ &= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{2}{j-i+1} \\ &= \sum_{i=1}^{n-1}\sum_{k=2}^{n-i+1}\frac{2}{k} \end{aligned}$$

The last inequality holds because we can change the limit of summation, by starting at $k=2$ since we add $i+1$, and then we also have to stop at $n-i+1$. This is a summation trick We can now upper bound this by extending the sum.

$$\begin{aligned} E\left[\sum_{i=1}^{n}\sum_{j>i} X_{ij}\right] &= \sum_{i=1}^{n-1}\sum_{k=2}^{n-i+1}\frac{2}{k} \leq \sum_{i=1}^{n}\sum_{k=2}^{n}\frac{2}{k} = \sum_{i=1}^{n} 2\cdot\sum_{k=2}^{n}\frac{1}{k} \\ &= 2n\cdot\sum_{k=2}^{n}\frac{1}{k} \\ &= 2n\cdot\left(\sum_{k=1}^{n}\frac{1}{k} - 1\right) \text{ first k=1 will make frac 1} \\ &= 2n(H_n - 1) \\ &= 2n\cdot H_n - 2n \\ &= O(n\log n) \end{aligned}$$

so we get in expectation randomized quicksort makes $O(n\log n)$comparisons.

3 Randomized min-cut

A multigraph is a graph that might contain multiple edges between pairs of vertices. A cut in a graph is, according to the book and in this setting, a set of edges whose removal causes the graph to be broken into two or more pieces. A min-cut is a cut of minimum cardinality(min number edges) that split the graph in at least two.

A very simple randomized algorithm is now presented for finding a min-cut.

1. Pick an edge uniformly at random.
2. Merge the endpoint vertices of the edge into 1 new vertex.
3. If there as a result of the contraction exists edges between vertices(including the new vertex), then keep them
4. All edges between the merged vertices are removed (we have a multigraph), so we have no self-loops to the new vertex.
5. When two vertices remain, the remaining edges between the vertices is a potential cut.

3.1 Contractions

Merging two endpoints of an edge into a new vertex, and removing any other edges between the two vertices to eliminate self-loops, is known as a contraction. This process reduces the vertex count by 1.

3.2 Lemma

RandMinCut returns a cut.

Proof

Let $V_1, V_2 \subset V$ be two subsets of V. Both are constructed from contraction of vertices. So this essentially the termination condition of the algorithm - all vertices have been contracted into two vertices V_1, V_2. The set of edges, C, going from $V_1$to V_2 are the proposed min-cut found by the algorithm. We will now prove that this is a cut in G.

For contradiction, suppose that C is not a cut in G. This means we have a path $\pi = u \rightsquigarrow v$ from a vertex $u \in V_1$ to $v \in V_2$ in G after removing the edges in C. In other words, one edge, e, on the path π that connects V_1 and V_2 is not in C. However, we removed edges in two ways

1. When we select an edge, and merge endpoint we remove the selected edge
2. Removing self-loops after merging

and in either way, the edges removed are between vertices that end in the same set - either V_1 or V_2. So the remaining edges must be those in C. This contradicts that e can exists and thereby that there is a path π from u to v. Therefore, all vertices that connect V_1 and V_2 are those in C, and therefore C is a cut, since removing the edges in C results in the graph being split up in the two sets V_1 and V_2.

3.3 Neighborhood

The neighborhood of a vertex, $N[v]$, is the set of vertices adjacent to v. The degree, $d(v)$, is the number of edges incident on v. For a set S, $N[S]$, is the union of the neighborhood of its constituents.

When there are no self-loop we have $N[v] = d(v)$.

3.4 Correctness of rand min-cut

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

Proof

First lets note that the degree $d(v) \geq |C|$ for all $v \in V$, since otherwise we could remove the edges incident on $v$and have a cut smaller than $|C|$ - contradicting C is a min-cut. Additionally, we get that

$$|E| = \frac{1}{2}\sum_{v \in V} d(v) \geq \frac{1}{2}n \cdot |C| \tag{1}$$

where the first inequality holds because each edge is counted twice (once for each of its endpoints) when calculating the degree over all vertices. The inequality holds by using the bound on the degree from earlier. Hence the number of edges is at least $\frac{1}{2}n \cdot |C|$. The proof is about bounding the probability of never contracting an edge in C.

Let A_i denote the event of not picking an edge in C at the i'th iteration of the algorithm for $i = 1, 2, ..., n-2$ (two vertices must remain at termination). During the first iteration of the algorithm, the probability of picking an edge from C is

$$\frac{|C|}{|E|} \leq \frac{|C|}{\frac{1}{2}n \cdot |C|} = \frac{2}{n}$$

where the first inequality holds since we know $|E| \geq \frac{1}{2}n \cdot |C|$, so the substitution makes the fraction larger. The probability of event A_1 is then bounded by $P(A_1) \geq 1 - \frac{2}{n}$. If A_1 happens, then what is the probability of not picking an edge from C agin? That is, what is $\Pr(A_2|A_1)$. Assuming A_1 was successful then the current graph must have at least

$$|E_1| \geq \frac{1}{2}(n-1) \cdot |C|$$

since we can again use (1) on the remaining $n-1$ vertices, since these must also have a degree of at least $|C|$, otherwise we could just remove the vertices incident to a vertex with $d(v) < |C|$ and we have smaller cut. Note this only holds because we only removed self loops in addition to the contracted edge. Hence we have that

$$\Pr(A_2|A_1) = 1 - \frac{|C|}{|E_1|} \geq 1 - \frac{|C|}{\frac{1}{2}(n-1) \cdot |C|} = \frac{2}{(n-1)}$$

and the probability is bounded by $\Pr(A_2|A_1) \geq 2/(n-1)$. Generally after the i'th iteration, we have removed i vertices, and the remaining vertices must have degree of at least $|C|$, so the graph must at this point have at least $\frac{1}{2}(n-i)\cdot|C|$ edges. Making the probability of not choosing an edge in C at the i'th iteration, given the previous $i-1$ previous iteration did not do so either is

$$\Pr\left(A_i|\cap_{j=1}^{i-1}A_j\right) \geq 1 - \frac{2}{(n-i+1)}$$

note the $(i+1)$ since at $i=2$ we had $n-1$. Each contraction reduces the vertex count by 1 and we continue until 2 vertices remain. So we want to bound the probability that no edge in C was picked in all $n-2$ iterations (this leaves 2 vertices left). This is different from $\Pr\left(A_i|\cap_{j=1}^{i-1}A_j\right)$ since we want it to hold simultaneously for all iterations. That is, we do not want a conditional probability, but we want

$$\Pr(\text{no edge in } C \text{ is ever picked by rand min-cut}) = \Pr\left(\cap_{i=1}^{n-2}A_i\right)$$

We can use the fact that

$$\Pr\left(\cap_{i=1}^{k}B_i\right) = \Pr(B_1)\cdot\Pr(B_2|B_1)\cdot,...,\cdot\Pr\left(B_k\cap_{i=1}^{k-1}B_i\right)$$

together with the upper bound $\Pr\left(A_i|\cap_{j=1}^{i-1}A_j\right)$ we have on each

$$\begin{aligned}
\Pr\left(\cap_{i=1}^{n-2}A_i\right) &\geq \prod_{i=1}^{n-2} 1 - \frac{2}{(n-i+1)} \\
&= \prod_{i=1}^{n-2} \frac{n-i+1}{n-i+1} - \frac{2}{(n-i+1)} \\
&= \prod_{i=1}^{n-2} \frac{n-i+1-2}{n-i+1} \\
&= \prod_{i=1}^{n-2} \frac{n-i-1}{n-i+1}
\end{aligned}$$

The denominator and numerator differ by 2, so we have a telescoping sum, where each term is 2 aspart. So the denominator will take values $n, n-1, ..., 4, 3$ and the numerator starts at $n-2, n-3, ..., 2, 1$. So everything cancels out except

$$\Pr\left(\cap_{i=1}^{n-2}A_i\right) \geq \frac{2\cdot 1}{n\cdot(n-1)} \geq \frac{2}{n^2}$$

and so with probability more than $\frac{2}{n^2}$ we will return a min-cut. So if we were to repeat the algorithm $\frac{n^2}{2}$times, where each time we choose an edge uniformly at random, the probability of not returning a min cut is

$$\prod_{i=1}^{n^2/2} 1 - \frac{n^2}{2} < 1/e$$

So we can control how accurate the algorithm is by just running it more times and sacrifice performance.