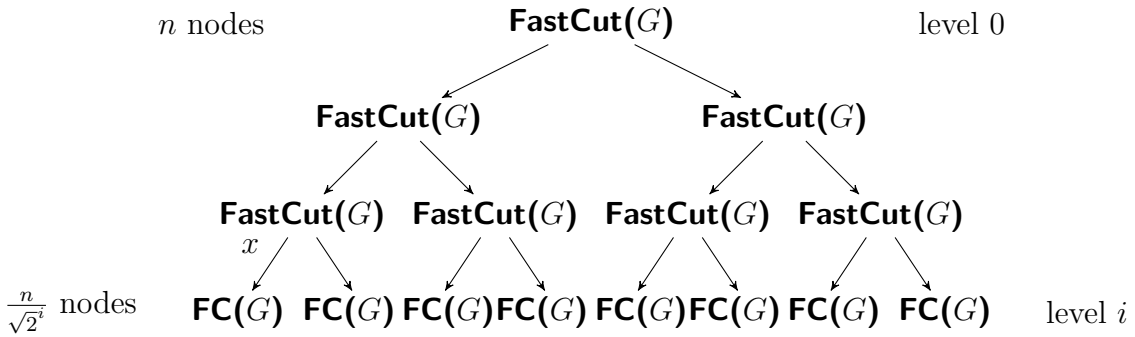
Theorem 1.17. *The algorithm* `FastCut` *always outputs a cut. With a probability of at least* $\Omega(1/\log n)$*, this cut is a minimum cut.*

Proof. As before, we fix an optimal solution S^* and let $C^* := \delta(S^*)$ be the edges that are cut by S^*. We show a lower bound on the probability that the `FastCut` algorithm does not output S^*.

The algorithm is recursive. Thus, the success probability follows a recurrence relation. Assume that we are at a call of `FastCut` on level i of the recursion, and C^* is still intact (no edge from C^* has been contracted). By level i of the recursion we mean all call where the input graph has already been reduced i times. The first call `FastCut(`G`)` is (the only call) on level 0.



We want to analyze the probability that a `FastCut` call on level i outputs an optimal solution, under the assumption that this is still possible. We let $P(i)$ be a lower bound on the probability that a call on level i outputs a minimum cut under the assumption that C^* is in its input graph.

`FastCut` computes a graph H_1 by calling `Contract(`G, t`)` with $t = 1 + \lceil n/\sqrt{2} \rceil$. We already argued that this call contracts no edge from C^* with probability at least $1/2$ because of the choice of t. Later, it recursively computes a solution S_1 based on H_1. The probability that H_1 still contains C^* and then the recursive call is successful (thus, $S_1 = S^*$) is at least $P(i+1)/2$ because these two events are independent. The same is true for the probability that S_2 is S^*. The probability that both S_1 and S_2 fail to be S^* is now bounded above by $(1 - P(i+1)/2)^2$, which means that

$$P(i) \geq 1 - \left(1 - \frac{P(i+1)}{2}\right)^2.$$

It might be a bit unintuitive that the recurrence goes 'up'. To change this, let h be the maximum level, i.e. the level where all calls are executed without any further recursion. We redefine $P(i)$ as the probability that a call on level $h - i$ outputs C^* under the assumption that C^* is in its input graph. Then $P(0) = 1$ because of the assumption that C^* is still a feasible solution and because the problem is solved optimally. Furthermore, by our above argumentation,

$$P(i) \geq 1 - \left(1 - \frac{P(i-1)}{2}\right)^2$$

for $i \in \{1, \ldots, h\}$.

We show that $P(i) \geq 1/(i+1)$ for all $i \in \{0, \ldots, h\}$ by induction. The base case is $i = 0$, and the statement holds in this case because $P(i) = 1 = 1/(0+1)$. For $i \in \{1, \ldots, h\}$ we get

$$\begin{aligned} P(i) \geq & 1 - \left(1 - \frac{P(i-1)}{2}\right)^2 \geq 1 - \left(1 - \frac{1}{2i}\right)^2 = 1 - \left(\frac{2i-1}{2i}\right)^2 \\ = & \frac{4i^2}{4i^2} - \frac{4i^2 - 4i + 1}{4i^2} = \frac{4i-1}{4i^2} = \frac{1}{i} - \frac{1}{4i^2} \overset{i \geq 1}{\geq} \frac{1}{i} - \frac{1}{i(i+1)} = \frac{1}{i+1}. \end{aligned}$$

The success probability of the first `FastCut` call is $P(h)$. On level i, the calls get input graphs with $n/\sqrt{2}^i$ nodes. Thus, on level $\lceil \log_{\sqrt{2}} n/6 \rceil$, the number of nodes is at most

$$\frac{n}{\sqrt{2}^{\lceil \log_{\sqrt{2}} n/6 \rceil}} \leq \frac{n}{n/6} = 6.$$

We conclude that $h \in \mathcal{O}(\log n)$ which proves the theorem. □

As before, we can additionally increase the success probability by independent repetitions. For a constant success probability, we need $\Theta(\log n)$ repetitions, yielding an overall running time of $\mathcal{O}(n^2 \log^2 n)$.

1.3.2 Reservoir Sampling

We now do an excursion to the data stream model to see a somewhat different setting. Data streams frequently occur in today's algorithmic challenges, when monitoring internet traffic or the output of a sensor, the stock market or another source that never stops to produce more and more information. Data streams cannot be stored (completely), and the length of the stream is unknown (or thought of as being infinite). In particular, this means that algorithm working with a data stream as the input data have no random access to the data. They can read the data once, store some information on it, but they cannot go back and forth.

There are many models for data stream algorithms. A detailed introduction and overview on different models is for example given in [4]. We do worst-case analysis in this part of the lecture, thus we also assume a worst-case scenario here: The data arrives in an arbitrary order, possibly the worst order for the algorithm we design. In this section, we consider the following simple data stream model: The stream consists of distinct integers

$$a_1, a_2, a_3, \ldots$$

in arbitrary order and we do not know the length of the stream (it might be infinite).

Sampling an element from a stream. In the data stream model, easy tasks can be a challenge. In this section, we want to solve the basic problem of choosing elements uniformly at random from the stream (choosing elements uniformly at random from some ground set is also called 'sampling'). More precisely, our first task is to design an algorithm that accomplishes the following: It always stores one number s from the stream. After seeing the ith element a_i, the probability that s is equal to a_j should be $1/i$ for all $j \in \{1, \ldots, i\}$. It is a bit surprising that we can actually achieve this goal, and in a fairly straightforward manner. Our algorithm does the following:

`ReservoirSampling`

1. After seeing a_1, set $s = a_1$.
2. When seeing a_i:
3. With probability $1/i$, replace s by a_i.
4. Else (thus, with probability $(i-1)/i$), do nothing.

Lemma 1.18. *`ReservoirSampling` satisfies for any $i \geq 1$: After seeing and processing a_i, it holds that*

$$\mathbf{Pr}(s = a_j) = \frac{1}{i} \quad \forall j \in \{1, \ldots, i\}.$$

Proof. The event '$s = a_j$' occurs if a_j is chosen to replace s after seeing a_j and it is never replaced by the $i-j$ subsequent elements (i.e. the algorithm chose to do nothing $i - j$ times). Thus,

$$\mathbf{Pr}(s = a_j) = \frac{1}{j} \cdot \frac{j}{j+1} \cdot \frac{j+1}{j+2} \cdot \frac{j+2}{j+3} \cdot \ldots \cdot \frac{i-2}{i-1} \cdot \frac{i-1}{i} = \frac{1}{i}$$

which proves the lemma. □

Sampling more than one element. Our second task is to sample s elements from the stream, where s can be greater than one. First we observe that this reduces to our previous algorithm if we allow that an element is sampled multiple times. In order to get a set S with s elements from the stream where multiples are allows, we simply run s copies of our above algorithm. Then every set S with $|S| = s$ has equal probability to be the current set after seeing a_i, namely probability $1/i^s$.

The challenge of this paragraph is to always have a set S of distinct elements which is chosen uniformly at random / sampled from the set of all possible subsets of $\{a_1, \ldots, a_i\}$. The algorithm that achieves this is the algorithm `R` published by Vitter in [9]. We still call the algorithm `ReservoirSampling` (for s elements):

ReservoirSampling

1. After seeing the first s elements, set $S = \{a_1, a_2, \ldots, a_s\}$.
2. When seeing a_i:
3. With probability $\frac{i-s}{i}$, do nothing.
4. Else (thus, with probability $\frac{s}{i}$):
5. Choose an item x uniformly at random from S.
6. Replace x in S by a_i.

We show that every set with s elements has equal probability to be S at any point in the stream. Observe that there are $\binom{i}{t}$ subsets of $\{a_1, \ldots, a_i\}$. Thus, after seeing a_i, the probability for any specific subset should be $1/\binom{i}{t}$.

Theorem 1.19. *`ReservoirSampling` (for s elements) satisfies for any $i \geq s$: After seeing and processing a_i, it holds that*

$$\mathbf{Pr}(S = T) = \frac{1}{\binom{i}{t}}$$

for every $T \subseteq \{a_1, \ldots, a_i\}$ with $|T| = s$.

Proof. We show the statement by induction over i.

$i = s$: Since the algorithm chooses the first s elements as the initial set S, we have $\mathbf{Pr}(S = T) = 1 = 1/\binom{1}{1}$ for $T = \{a_1, \ldots, a_s\}$, the only possible subset of $\{a_1, \ldots, a_s\}$ that satisfies $|T| = s$.

$i > s$: We want to show $\mathbf{Pr}(S = T) = 1/\binom{i}{t}$ for all possible sets, so let $T \subset \{a_1, \ldots, a_i\}$ be an arbitrary fixed set. We let S' be the set that the algorithm had after seeing a_{i-1} and before seeing a_i. It then processed a_i, maybe replaced it by a uniformly chosen element x and the current set is set S. We analyze the probability that $S = T$ by distinguishing two cases.

a) $a_i \notin T$.
If a_i is not in T, then the algorithm can only have $S = T$ if it already had T after seeing a_{i-1}. Thus let A be the event that $S' = T$. By the induction hypothesis, we know that

$$\mathbf{Pr}(A) = \frac{1}{\binom{i-1}{s}}.$$

If A occurred, it might still be that $S \neq T$ if the algorithm chose to replace an item. Thus let B be the event that the algorithm replaces an item. By the algorithm definition, it is

$$\mathbf{Pr}(B) = \frac{s}{i}.$$

We observe that $S = T$ occurs exactly if A occurs and B does not occur. Since the events A and $\bar{B}$ are independent, we can conclude that

$$\mathbf{Pr}(S = T) = \mathbf{Pr}(A \cap \bar{B}) = \mathbf{Pr}(A) \cdot \mathbf{Pr}(\bar{B})$$

$$= \frac{1}{\binom{i-1}{s}} \cdot \frac{i-s}{i} = \frac{s!(i-s-1)!}{(i-1)!} \cdot \frac{i-s}{i} = \frac{s!(i-s)!}{i!} = \frac{1}{\binom{i}{s}}.$$

b) $a_i \in T$.
In this case, we get T if we had all elements except a_i in S' already. Thus, let $R = T \backslash \{a_i\}$ be these elements and let C be the event that $R \subset S'$. Notice that there are $(i-1)-(s-1) = i-s$ subsets of $\{a_1, \ldots, a_{i-1}\}$ with $s-1$ elements that contain R. For each of these subsets, the probability that it is S' is $1/\binom{i-1}{s}$. Thus we get that

$$\mathbf{Pr}(C) = \frac{i-s}{\binom{i-1}{s}}.$$

We again use the event B that the algorithm chose to replace an item by a_i, we recall that $\mathbf{Pr}(B) = s/i$. Finally, we need one event more. Let D be the event that $C \cap B$ already occurred and additionally, the item x is the one item in $T \backslash R$. Thus, $\mathbf{Pr}(S = T) = \mathbf{Pr}(D)$. Observe that the probability $\mathbf{Pr}(D \mid B \cap C)$ is $1/s$: If we already know that an element is replaced, and we also already know that C occurred, then the probability that the replaced item is one specific item is just $1/|S| = 1/s$. We can now conclude that

$$\begin{aligned}
\mathbf{Pr}(S = T) &= \mathbf{Pr}(B \cap C \cap D) = \mathbf{Pr}(B \cap C) \cdot \mathbf{Pr}(D \mid B \cap C) \\
&= \mathbf{Pr}(B) \cdot \mathbf{Pr}(C) \cdot \mathbf{Pr}(D \mid B \cap C) \\
&= \frac{s}{i} \cdot \frac{i-s}{\binom{i-1}{s}} \cdot \frac{1}{s} \\
&= \frac{1}{i} \cdot \frac{i-s}{\binom{i-1}{s}} \\
&= \frac{1}{\binom{i}{s}}.
\end{aligned}$$

For the last step, observe that we computed the same term above as well. We have shown that the theorem is true.

□