Skip lists: A randomized dictionary

The exposition is based on the following sources, which are all recommended reading:

1. Pugh: Skip lists: a probabilistic alternative to balanced treed. Proceedings WADS, LNCS 382, 1989, pp. 437-449
2. Sedgewick: Algorithmen in C++, 2002, Pearsons, (Chapter 13.5)
3. Lecture Script from Michiel Smid, University of the Saarland.
4. Motwani, Raghavan: Randomized algorithms, Chapters 8.3 and 4.1
5. Kleinberg, Tardos: Algorithm design, Chapter 13.9

Introduction

Here a little refresher of sum formulas you will need:

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k \cdot y^{n-k}$$

and for $0 < r < 1$:

$$\sum_{k=0}^{n} r^k = \frac{1-r^{n+1}}{1-r}$$

$$\sum_{k=0}^{\infty} r^k = \frac{1}{1-r}$$

Introduction

We consider the so called *dictionary problem*. Given a set S of real numbers, store them in a data structure such that the following three opertions can be performed efficiently:

- **Search(x):** Given the real number x, report the maximal element of $S \cup \{-\infty\}$ that is at most equal to x.
- **Insert(x):** Given a real number x insert it into the data structure.
- **Delete(x):** Given a real number x delete it from the data structure.

The standard data structures for this problem is the balanced binary tree. It supports all the above operations in worst case time $O(\log n)$ and uses $O(n)$ space.

Well known classes of balanced trees are for example AVL-trees, BB[α]-trees and red-black-trees. In order to maintain their worst case time behaviour all those data structures need more or less elaborate *rebalancing* operations which make an implementation non-trivial, which in turn leads not to the best practical run times.

We will introduce in this lecture an alternative, *randomized* data structure, the *skip list*. It uses *in expectation* linear space and supports the above dictionary operation in *expected* time $O(\log n)$ with *high probability*.

Why do we do this? We will see that the data structure is conceptually much simpler and more elegant than balanced trees. Nevertheless we will exchange a worst case runtime against an expected run time.

However, the analysis will show that skip lists behave very well and are very fast in practice (the difference is similar to the deterministic merge sort and the randomized quicksort algorithm).

The goal of this lecture is to

- introduce you to the data structure
- show you how to analyze the randomized run time
- introduce you to *tail estimates* using *Chernoff bounds*.

Skip lists

Throughout the lecture we assume that we can generate random, independent bits in unit time. Let S be a set of n real numbers. Then we construct a sequence of sets $S_1, S_2, ...$ as follows:

1. For each element $x \in S$, flip a coin until zero comes up.
2. For each $i \geq 1$, S_i is the set of elements in S for which we flipped the coin at least i times.

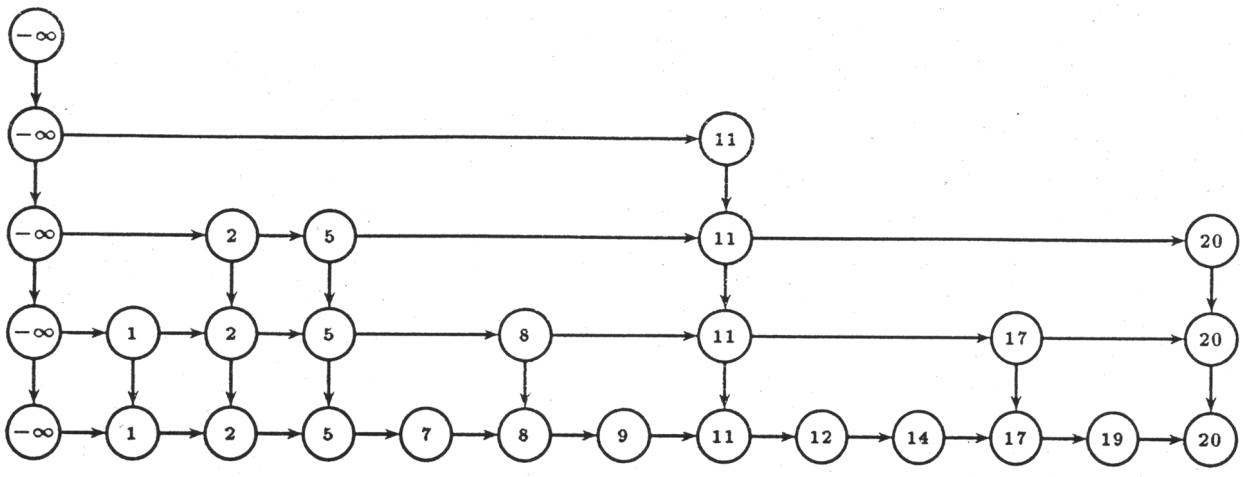
Let h be the number of sets that are constructed. Then it is clear that

$$\emptyset = S_h \subseteq S_{h-1} \subseteq S_{h-2} \subseteq \cdots \subseteq S_2 \subseteq S_1 = S$$

The skip list for S consists then of the following:

1. For each $1 \leq i \leq h$, the elements of $S_i \cup \{-\infty\}$ are stored in a sorted linked list L_i.
2. For each $1 < i \leq h$, there is a pointer from each $x \in L_i$ to its occurrence in L_{i-1}.

Here is an example. Suppose $S = \{1,2,5,7,8,9,11,12,14,17,19,20\}$. Flipping coins might lead to $S_1 = S$, $S_2 = \{1,2,5,8,11,17,20\}$, $S_3 = \{2,5,11,20\}$, $S_4 = \{11\}$, and $S_5 = \emptyset$.



Searching skip lists

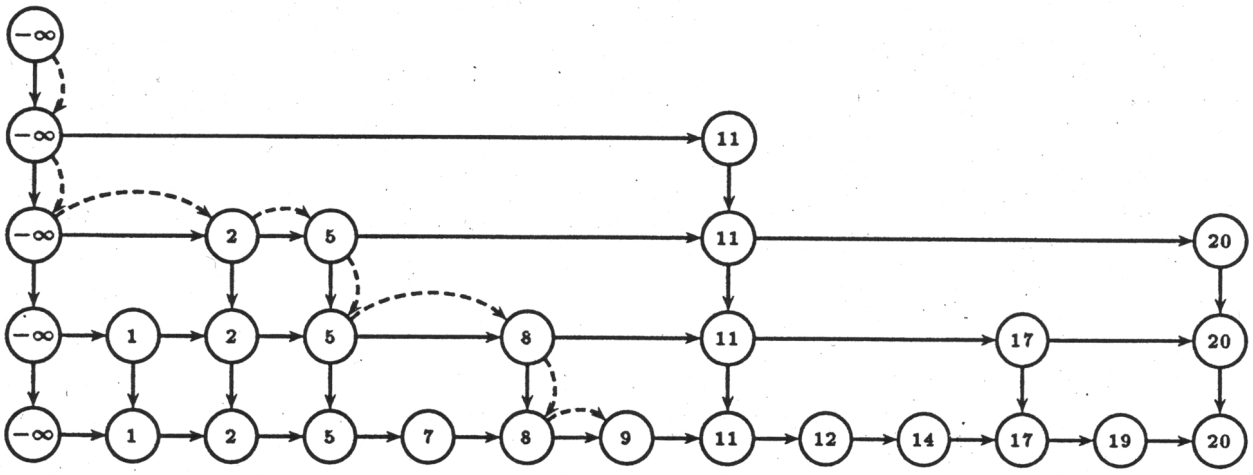
We can now implement the search for x as follows:

1. Let y_h be the only element in L_h.
2. For $i = h, h-1, ..., 2$
   (a) Follow the pointer from y_i in L_i to its occurence in L_{i-1}.

(b) Starting in y_{i-1}, walk to the right along L_{i-1}, until an element is reached that is larger than x or the end of L_{i-1} is reached. Let now y_{i-1} be the last encountered element in L_{i-1} that is at most equal to x.

3. Output y_1.

The following figure illustrates the search step (we search for element 10):

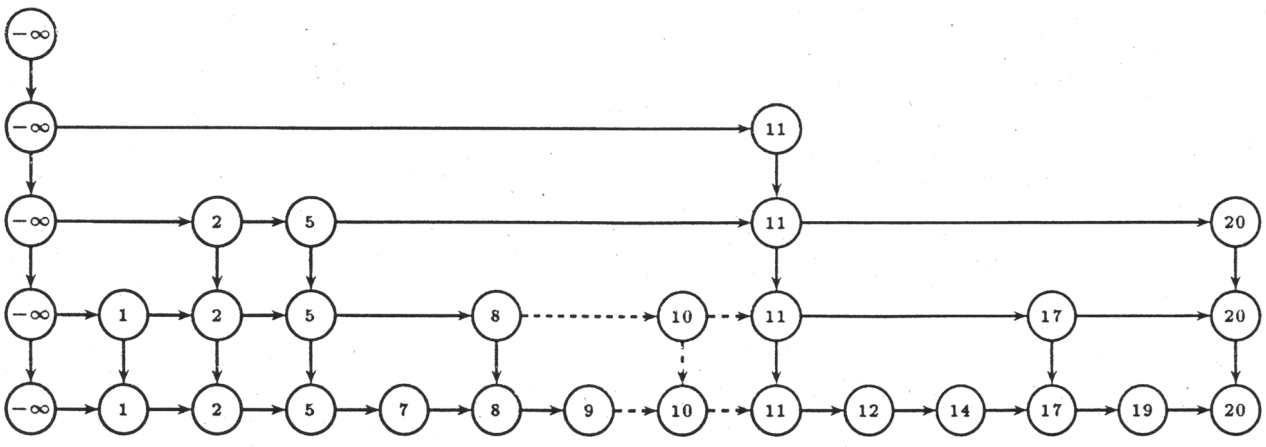


It is not hard to imagine how the insertion and deletion operations work on a skip list.

Inserting into a skip list

For inserting an element x into the dictionary we proceed as follows:

1. Run the search algorithm for x. Let $y_1, y_2, \ldots, y_h$ be the elements of $L_1, L_2, \ldots, L_h$ that are computed while searching. If $x = y_1$, the $x \in S$ and nothing has to be done. Hence assume that $x \neq y_1$.
2. Flip a coin until a zero comes up. Let l be the number of coin flips.
3. For each $1 \leq i \leq \min(l, h)$, add x to the list L_i immediately after y_i.
4. If $l \geq h$, then create new lists $L_{h+1}, \ldots, L_{l+1}$ storing the sets $S_{h+1} \cup \{-\infty\}$, where each set contains x except for S_{l+1} which is empty.
5. For each $1 < i \leq l$, give x in L_i a pointer to its occurrence in L_{i-1}.
6. If $l \geq h$, then for each $h+1 \leq i \leq l+1$, give $-\infty$ in L_i a pointer to its occurrence in L_{i-1}.
7. Set $h = \max(h, l+q)$.



Deleting from a skip list

1. Run the search algorithm for x. Let $y_1, y_2, \ldots, y_h$ be the elements of $L_1, L_2, \ldots, L_h$ that are computed while searching. If $x \neq y_1$, the $x \notin S$ and nothing has to be done. Hence assume that $x = y_1$.
2. For each $1 \leq i \leq h$ such that $x = y_i$ delete y_i from the list L_i.
3. For $i = h, h-1, \ldots$: if L_{i-1} only stores $-\infty$, delete the list L_i and set $h = h-1$.

Why are skip lists efficient? The intuition

We have seen, that mostly what we do in skip lists is to search. The rebalancing is done by throwing a coin a few times and making local changes along the search path.

How expensive is the search? It is the sum over all traversed path length at each level. We expect there to be $\approx \log n$ levels. At each level we travel to the right. However for a fixed level we do not expect to do this long, since this would imply, that all the elements are *not* in the level above.

Hence we expect to spend a constant amount of time at each level which would add up to a total search time of $O(\log n)$. We will now prove this more formally.

Why are skip lists efficient? The proofs

The size of a skip list and the running times of the search and update algorithms are random variables. We will prove that their expected values are bound by $O(n)$ and $O(\log n)$ respectively.

Recall that h denotes the number of sets S_i that result from our probabilistic construction. How can we derive an upper bound for h?

Let x be an element of S and $h(x)$ be the number of sets S_i that contain x. Then $h(x)$ is a random variable distributed acccording to a geometric distribution with $p = 1/2$. Hence $Pr(H(x) = k) = (1/2)^k$ and $E(h(x)) = 2$. That means if we look at a specific element we only expect it to be in S_1 and S_2.

Clearly $h = 1 + \max\{h(x) : x \in S\}$. From $E(h(x)) = 2$ for any $x \in S$, however, we cannot conclude that the expected value of h is three.

We can estimate $E(h)$ as follows. Again consider a fixed $x \in S$. It follows that for any $k \geq 1$, $h(x) \geq k$ if and only if the first $k-1$ coin flips produced a one. That is $Pr(h(x) \geq k = (1/2)^{k-1}$. In addition it is clear that $h \geq k+1$ if and only if there is a $x \in S$ such that $h(x) \geq k$. Hence

$$Pr(h \geq k+1) \leq n \cdot Pr(h(x) \geq k) = \frac{n}{2^{k-1}}$$

This estimate does not make sense for $k < 1 + \log n$. For those values of k we can use the trivial upper bound $Pr(h \geq k+1) \leq 1$. Then $E(h)$ equals:

$$\sum_{k=0}^{\infty} Pr(h \geq k+1) = \sum_{k=0}^{\lceil \log n \rceil} Pr(h \geq k+1) + \sum_{k=1+\lceil \log n \rceil}^{\infty} Pr(h \geq k+1).$$

(exercise: proof the first equality, that is $E(X) = \sum_{k=1}^{\infty} Pr(X \geq k)$ for a random variable X that takes values $\{0, 1, 2, \ldots\}$.)

The first summation on the right hand side is at most $1 + \lceil \log n \rceil$. The second sum can be bounded from above by:

$$\sum_{k=1+\lceil \log n \rceil}^{\infty} \frac{n}{2^{k-1}} = n(1/2)^{\lceil \log n \rceil - 1} \leq n(1/2)^{\log n - 1} \leq 2.$$

Hence we have proven that $E(h) \leq 3 + \lceil \log n \rceil$.

The expected size of a skip list can easily be computed. Let M denote the total size of the sets $S_1, S_2, \ldots, S_h$. Then $M = \sum_{x \in S} h(x)$ and by linearity of expectation:

$$E(M) = \sum_{x \in S} E(h(x)) = \sum_{x \in S} 2 = 2n.$$

If M' denotes the total number of nodes in a skip list, then M' is equal to M plus h. Hence

$$E(M') = E(M + h) = E(M) + E(h) \leq 2n + 3 + \lceil \log n \rceil.$$

What is left to do is to estimate the search costs.

Let x be a real number and let C_i denote the number of elements in the list L_i that are inspected when searching for x (We do not count the element of L_i at which the algorithm starts walking to the right. Hence, C_i counts comparisons between x and elements of S.) The search cost is then proportional to $\sum_{i=1}^{h}(1 + C_i)$.

Again we cannot use linearity of expectation since h is a random variable. Again the trick is to fix an integer A and analyze the search cost up to a level A and above level A separately (and differently).

We first estimate the search level above A, i.e., the total costs in the lists $L_{A+1}, L_{A+2}, \ldots, L_h$. Since the cost is at most equal to the total size of these lists, its expected value is at most equal to the expected value of $M_A := \sum_{i=A+1}^{h} \mid L_i \mid$.

How do we estimate this value? We first note that the lists L_i, $A+1 \leq i \leq h$, form a skip list for S_{A+1}. Hence we have:

$$E(M_A) = \sum_{k=0}^{n} E(M_A \mid\mid S_{A+1} \mid = k) \cdot Pr(\mid S_{A+1} \mid = k)$$

where $E(M_A \mid\mid S_{A+1} \mid = k)$ is the expected size of a skip list with k elements. We have already seen that this is $O(k)$.

Hence we only need to compute $Pr(\mid S_{A+1} \mid = k)$. Since $\mid S_{A+1} \mid = k$ if and only if out of the n elements of S exactly k reach the level $A+1$, we have:

$$Pr(\mid S_{A+1} \mid = k) = \binom{n}{k} (\frac{1}{2})^{Ak} (1 - (\frac{1}{2})^{A})^{n-k}.$$

Setting $p = \frac{1}{2}^{A}$, we infer that the expected value of M_A is proportional to:

$$\begin{aligned}
\sum_{k=0}^{n} k \cdot \binom{n}{k} p^k (1-p)^{n-k} &= \sum_{k=1}^{n} n \cdot \binom{n-1}{k-1} p^k (1-p)^{n-k} \\
&= n \cdot p \sum_{k=0}^{n-1} \binom{n-1}{k} p^k (1-p)^{n-1-k} \\
&= n \cdot p (p + (1-p))^{n-1} \\
&= n \cdot p
\end{aligned}$$

Hence the expected search cost above level A is bounded by $O(n/2^A)$.

Next we estimate the expected search cost in the lists $L_1, L_2, \ldots, L_A$. Recall that C_i is the number of elements searched when searching for x. We use again conditional expectation. Let $l_i(x)$ be the number of elements in L_i that are at most equal to x. Then

$$E(C_i) = \sum_{k=1}^{n} E(C_i \mid l_i(x) = k) \cdot Pr(l_i(x) = k).$$

Assume that $l_i(k) = k$. Also assume that there is an element in L_i that is larger than x.

Then $C_i = j$ if and only if the largest $j-1$ elements of L_i that are at most equal to x do not appear in L_{i+1}, but the element that immediately precedes these $j-1$ elements does appear in L_{i+1}.

Hence

$$Pr(C_i = j \mid l_i(x) = k) \leq (\frac{1}{2})^{j-1},\ 0 \leq j \leq k.$$

This inequality also holds if x is at least equal to the maximal element of L_i. From this we obtain:

$$\begin{aligned} E(C_i \mid l_i(x) = k) &= \sum_{j=0}^{k} j \cdot Pr(C_i = j \mid l_i(x) = k) \\ &\leq \sum_{j=0}^{k} \frac{j}{2^{j-1}} \\ &\leq 4. \end{aligned}$$

(exercise. Hint: write the sum $\sum_{j=0}^{k} j \cdot x^{j-1}$ as a derivative of $\sum_{j=0}^{k} x^j$, apply bounding)

This, in turn implies that

$$E(C_i) \leq \sum_{k=1}^{n} 4 \cdot Pr(l_i(x) = k) = 4$$

It follows that the expected search cost up to level A is proportional to:

$$E(\sum_{i=1}^{A}(1 + C_i)) = \sum_{k=1}^{A}(1 + E(C_i)) \leq 5A$$

Summarizing we have shown that the expected search time for element x is bounded by:

$$O(\frac{n}{2^A} + A).$$

Setting A to $\log n$ we obtain the required bound of $O(\log n)$.

Tail estimates: Chernoff bounds

So far we proved bounds on the expected size, search time and update time for a skip list. In this section we conder so called *tail estimates*.

That is, we estimate the probability that the actual search time deviates significantly from its expected value. For example assume for a moment that the constant in the $O(\log n)$ term for the search time is one. Then we want to estimate the probability that the actual search time is at least $t \cdot \log n$.

We could derive an estimate using *Markov's inequality*.

Lemma 1. *Let X be a random variable that takes non-negative values, and let μ be the expected value of X. Then for any $t > 0$, $Pr(X \geq t\mu) \leq \frac{1}{t}$.*

Proof: Let $s = t\mu$. Then

$$\begin{aligned} \mu &= \sum_{x} x \cdot Pr(X = x) \qquad (3.1) \\ &\geq \sum_{x \geq s} x \cdot Pr(X = x) \\ &\geq \sum_{x \geq s} s \cdot Pr(X = x) \\ &= s \cdot Pr(X \geq s) \end{aligned}$$

Hence the probability that the actual search time is at least $t \cdot \log n$ is less than or equal to $1/t$.

This is not very impressive. The probablity that the search time is more than 100 times its expected value is at most $1/100$. So if this bounds was tight one search in a hundred takes more than 100 times the time of the average search.

In this section we will see that Chernoff bounds give a much tighter estimate. We will prove that the probablity that the search time exceeds $t \cdot \log n$ is less than or equal to $\approx n^{-t/8}$ for $t \geq 5$.

Hence in a skip list of 1000 elements, the probability that the search time is more than 100 times its expected value is 10^{-38} which in practice means, it will never occur. (Even for $t = 50$ the bound is still 10^{-19}, and for $t = 10$ the probability is still only $\approx 2 \cdot 10^{-4}$).

Markov's inequality holds for any non-negative random variable. The Chernoff technique applies to random variables X that can be written as the sum $\sum_{i=1}^{n} X_i$ of mutually independent random variables X_i.

(Variables are called (mutually) independent if their joint density function is the product of the individual density functions. Beware that mutual independence is different than pairwise independence! (exercise)).

In such cases much better bounds can be obtained.

So let $X_1, X_2, X_3 \ldots, X_n$ be a sequence of mutually independent random variables and let $X = \sum_{i=1}^{n} X_i$.

The *moment generating function (mgf)* for a (discrete) random variable Y is defined as

$$m_Y(\lambda) = E(e^{\lambda Y}) = \sum_y e^{\lambda y} \cdot Pr(Y = y)$$

As the name suggests the function is used to easily generate the moments of the random variable Y. Clearly $m(0) = 1$ and it is easy to show that $\mu = m'(0)$ and $\sigma^2 = m''(0) - \mu^2$ (exercise).

In the case of X, which is a sum of n independent variables, $m_X(\lambda) = \prod_{i=1}^{n} m_{X_i}(\lambda)$ and of course the mean value of X is the derivate of the mgf at position 0 which is simply the product of all means of the X_i.

Or written down:

$$E(e^{\lambda X}) = E(e^{\lambda(X_1 + \cdots + X_n)}) = \prod_{i=1}^{n} E(e^{\lambda X_i}).$$

Now let $s > 0$ and $\lambda > 0$. Since $X \geq s$ if and only if $e^{\lambda X} \geq e^{\lambda s}$, we have $Pr(X \geq s) = Pr(e^{\lambda X} \geq e^{\lambda s})$. By applying Markov's inequality to the non negative random variable $e^{\lambda X}$, we get

$$Pr(X \geq s) = Pr(e^{\lambda X} \geq e^{\lambda s}) \leq e^{-\lambda s} \cdot E(e^{\lambda X}).$$

This yields:

$$Pr(X \geq s) \leq e^{-\lambda s} \cdot \prod_{i=1}^{n} E(e^{\lambda X_i}), \text{ for } s > 0 \text{ and } \lambda > 0.$$

This is the *basic inequality* we work with. To estimate $Pr(X \geq s)$ we need bound on $E(e^{\lambda X_i})$. Of course those bounds depend on the probability distribution of X_i. We will now illustrate the technique using the *geometric distribution* with parameter $p = 1/2$.

Let T be the number of flips we need until a one comes up in a series of coin flips. Then $Pr(T = k) = (1/2)^k$ for $k \geq 1$ and $E(T) = 2$. Now assume we are interested in T_n which is the number of flips we need until we obtain a one exactly n times (i.e. $T = T_1$).

If we define the random variables X_i as the number of flips between the $(i-1)-$st (excluding) and the i-th one (including), then X_i is distributed according to a geometric distribution. (This property is also called the memoryless property of the geometric or exponential distribution).

Then $T_n = \sum_{i=1}^{n} X_i$, where each X_i is distributed according to a geometric distribution and the expected value is $E(T_n) = 2n$, and Markov's inequality gives $Pr(T_n \geq (2+t)n \leq \frac{2}{2+t})$.

For $0 < \lambda < \log 2$ we have

$$E(e^{\lambda X_i}) = \sum_{k=1}^{\infty} e^{\lambda k} \cdot Pr(X_i = k) = \sum_{k=1}^{\infty} (e^{\lambda}/2)^k = \frac{e^{\lambda}}{2 - e^{\lambda}}$$

We now apply our basic inequality with $s = (2+t)n$, where $t > 0$ and get

$$Pr(T_n \geq (2+t)n) \leq e^{-\lambda(2+t)n}(\frac{e^{\lambda}}{2-e^{\lambda}})^n = (\frac{e^{-\lambda(1+t)}}{2-e^{\lambda}})^n$$

Now we choose λ such that the term on the right hand side is minimized (exercise) and find $\lambda = \log(1+\frac{t}{2+t})$.

Hence we have

$$Pr(T_n \geq (2+t)n) \leq (1+t/2)^n(1-\frac{t}{2+2t})^{(1+t)n}.$$

Since $1-x \leq e^{-x}$ for all x, we have

$$(1-\frac{t}{2+2t})^{1+t} \leq (e^{\frac{-t}{2+2t}})^{1+t} = e^{-t/2}.$$

Moreover, $1+t/2 \leq e^{t/4}$ for $t \geq 3$. This proves that for $t \geq 3$

$$Pr(T_n \geq (2+t)n) \leq e^{tn/4} \cdot e^{-tn/2} = e^{-tn/4}.$$

Compare this with the bound obtained from Markov's inequality (which was $\frac{2}{2+t}$)!

We can subsume our finding in the following theorem:

Theorem 2. *Let $X_1, X_2, \ldots, X_n$ be mutually indpendent random variables and assume that each X_i is distributed according to a geometric distribution. Let $T_n = \sum X_i$, then $E(T_n) = 2n$ and for any $t \geq 3$ holds:*

$$Pr(T_n \geq (2+t)n) \leq e^{tn/4} \cdot e^{-tn/2} = e^{-tn/4}.$$

Corollary 3. *Let $c \geq 1$ be a constant and let m be a positive integer. Further let $n = c \cdot \ln m$. Then for any $s \geq 5$ it holds*

$$Pr(T_{\lceil n \rceil} \geq sn) \leq m^{\frac{-(s-2)c}{4}}.$$