Probability for Computer Science

Prof. Nitin Saxena
Department of Computer Science and Engineering
Indian Institute of Technology - Kanpur

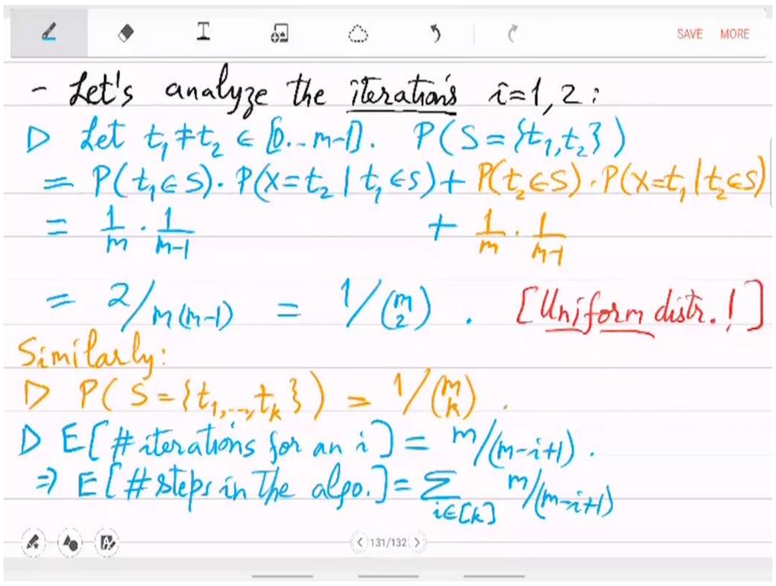
Module - 7
Lecture - 25
Biased Coin Tosses, Hashing

Last time we did many examples which are used in computer science, related to probability; so, very basic questions like, how do you sample a number from a range 0 to m - 1, where m can be anything. So, then, you saw that we basically reduce it to log m coin tosses. And we keep tossing until we get a number in this range 0 to m - 1, and not m or more. And we saw that this actually is a very fast algorithm.

You only need to toss 2 log m coins, or the same coin 2 log m times. That is the expectation. We did this by the geometric distribution. So, the lesson you learnt here is that expected number of steps here came out to be 1 over the probability, reciprocal of the probability, which is 2 raised to k by m. And then, using this, we can actually also sample a subset of size k. So, subset means that you want distinct numbers from the set S.

So, this was quite similar. And we saw that first 2 elements that it gives you, t 1, t 2; so, the probability of getting t 1, t 2 where t 1, t 2 are different, this comes out to be 1 over m choose 2. So, which means that amongst all possibilities, this is, all the possibilities have the same probability of turning up.

(Refer Slide Time: 01:53)



- Let's analyze the iterations $i=1,2$:

▷ Let $t_1 \neq t_2 \in [0..m-1]$. $P(S=\{t_1,t_2\})$

$= P(t_1 \in S)\cdot P(x=t_2 \mid t_1 \in S) + P(t_2 \in S)\cdot P(x=t_1 \mid t_2 \in S)$

$= \frac{1}{m}\cdot\frac{1}{m-1} + \frac{1}{m}\cdot\frac{1}{m-1}$

$= 2/m(m-1) = 1/\binom{m}{2}$. [Uniform distr.!]

Similarly:

▷ $P(S=\{t_1,\ldots,t_k\}) = 1/\binom{m}{k}$.

▷ $E[\#\text{iterations for an } i] = m/(m-i+1)$.

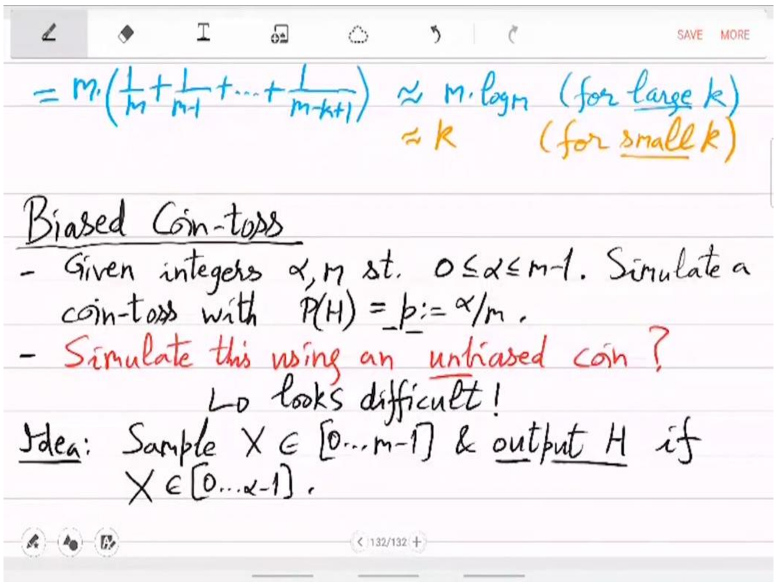
$\Rightarrow E[\#\text{steps in the algo.}] = \sum_{i\in[k]} m/(m-i+1)$

So, similarly you can get that probability of S being a subset or the subset t 1 to t k, this probability is really 1 over m chose k; so, which is really uniform probability over all the k subsets. And then, of course, you are interested in the number of times this GoTo happens, this mistake happens. So, this seems to be an infinite loop again. So, when is this seemingly infinite loop going to stop? What is the expectation of the number of attempts?

So, that will come out to be; so, expected number of iterations in this for loop for an i; so, that comes out to be again like the geometric distribution we did last time. It is 1 over the probability; so, it comes out to be m over m - i + 1; the ith step; i - 1 have been picked, elements have been picked; so, you want m - i + 1 possibilities. So, that over m is the chance of correct x. So, it is this.

And then, you sum it up over all i. So, this means that expected number of steps in the algo is Sigma i 1 to k and m over m - i + 1, so, which is m comes out; you have 1 over m + 1 over m - 1 + 1 over m - k + 1.

(Refer Slide Time: 04:13)



$= m\cdot\left(\frac{1}{m}+\frac{1}{m-1}+\dots+\frac{1}{m-k+1}\right) \approx m\cdot\log m$ (for large k)

$\approx k$ (for small k)

Biased Coin-toss

- Given integers α, m st. $0 \le \alpha \le m-1$. Simulate a coin-toss with $P(H) = p := \alpha/m$.
- Simulate this using an unbiased coin?

$\hookrightarrow$ looks difficult!

Idea: Sample $X \in [0 \dots m-1]$ & output H if $X \in [0 \dots \alpha-1]$.

So, what is the sum? If k is very close to m, then this is really the harmonic series. So, this is actually equal to m times log m for large k; otherwise it is different. Otherwise, if the k is very small, then this is like 1 over m; so, you get constant. So, for small k, this is like k steps in the algorithm, which is similar to picking 1 element. But if you are picking many, many elements, so, k is large; then it comes out to be m times log m; but either way it is quite small.

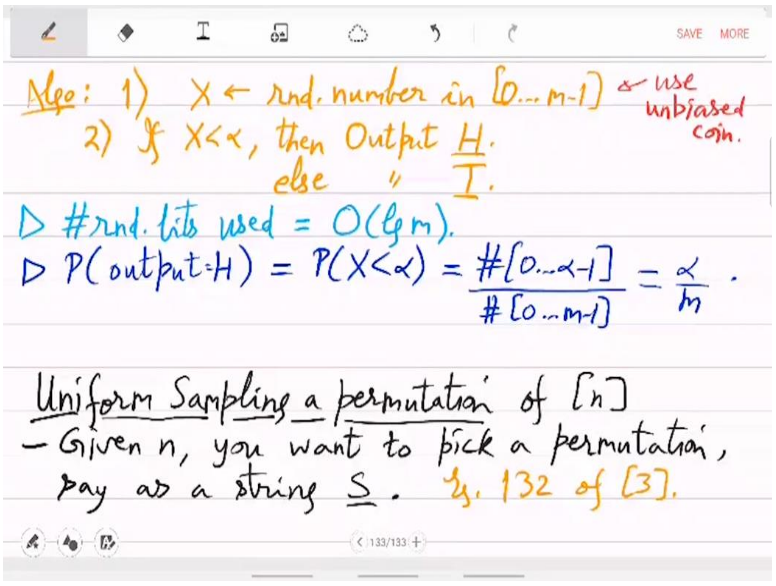
So, where are we now? So, you have learnt how to sample 1 element, k elements. And now, we will learn how to sample, how to maybe toss, simulate a biased coin toss. Let us do that. So, biased coin toss: So, the problem here is that, using a fair coin, an unbiased coin, you want to simulate a biased coin. This seems quite difficult actually. So, given integers alpha m such that alpha is between 0 and m - 1, simulate a coin toss with probability of heads being p which is alpha by m.

So, that is the problem. So, you know that unbiased coin automatically gives you half probability; but what if you do not want half, you want one-third or you want 10 over 11? So, how do you achieve those probabilities using a fair coin? So, simulate this using an unbiased coin. Can this be done? So, this actually looks difficult, because, how can you break this unbiased coin and make it crooked?

Seems to really require you to change the material of the coin; but, okay; of course, we will give a mathematical solution for this; we will not break the coin. So, the idea is to sample X in 0 to m - 1 and output heads if X is in the range 0 to alpha minus 1. So, you have seen this algorithm, how to sample an element in the range 0 to m - 1. This solution is just 1 step ahead of that. So, first you sample in the range 0 to m - 1.

And if X comes out to be actually in the sub-range 0 to alpha minus 1, then you output head; otherwise you output tail. And you can see why this makes sense, because we intend to show that the probability of X falling in the 0 to alpha minus 1 window is alpha by n. Hence, this is like a biased coin toss. So, let us write down the algorithm.

(Refer Slide Time: 09:09)



Algo: 1) $X \leftarrow$ rnd. number in $[0 \dots m-1]$ ← use unbiased coin.

2) If $X < \alpha$, then Output $\underline{H}$.
else 〃 $\underline{T}$.

▷ #rnd. bits used $= O(\lg m)$.

▷ $P(\text{output}=H) = P(X<\alpha) = \frac{\#[0 \dots \alpha-1]}{\#[0 \dots m-1]} = \frac{\alpha}{m}$.

Uniform Sampling a permutation of $[n]$

– Given n, you want to pick a permutation, say as a string $\underline{S}$. Eg. 132 of [3].

So, X is a random number in 0 to m - 1. This you can do by unbiased coin tosses. That will give you X in the range 0 to m - 1, uniformly distributed. And once you have this, you check whether X is less than alpha. If it is, then you have to output heads; else you output tails. So, since X is a random number in the range 0 to m - 1 generated by using fair coin, you know that the chance of outputting head is alpha by m. That is it.

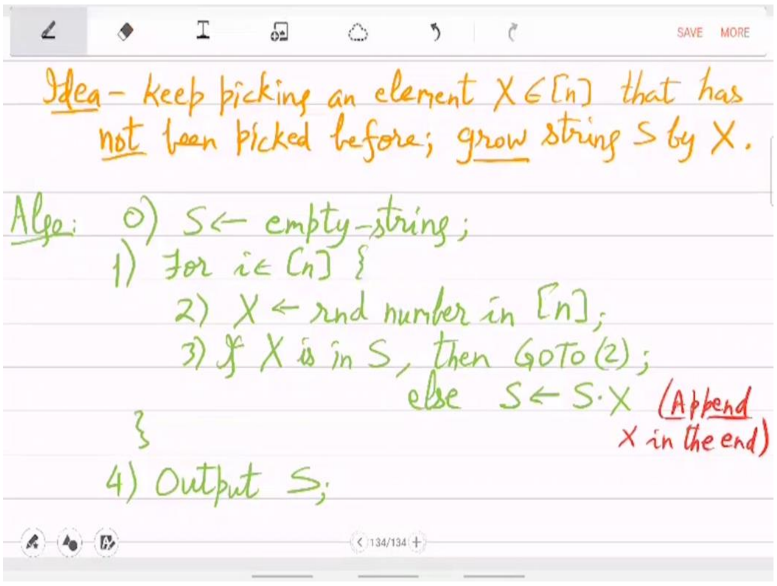
So, number of random bits used is only log m. And the probability of getting head is the same probability as this random variable coming out to be in the range 0 to alpha minus 1, which is the size of an interval 0 to alpha minus 1 divided by the size of 0 to m - 1, which is alpha by m. That finishes the implementation. So, that is about biased coin toss. And the final thing we will do is uniform sampling a permutation of elements 1 to n.

This too, you need in many applications when you have to; when you like to have a random permutation, then you need this algorithm. Now, what is this to do with coin tosses? How do you get a random permutation? Notice that this is somehow similar to sampling k elements.

There you are picking 1 element at a time and till you get k distinct elements. So, the similar idea you can use here.

And since there are n factorial permutations, you want a permutation to appear with probability 1 over n factorial. So, that is what we have to analyse; but let us first see the algorithm. So, given n, you want to pick a permutation, say as a string S. Permutation, we see as a string S. So, like 1 2 3 or 3 2 1; these are permutations of the set of 3 elements or 2 1 3. So, see it as a string. The idea is as I just said.

(Refer Slide Time: 13:15)



Idea - keep picking an element $X \in [n]$ that has not been picked before; grow string S by X.

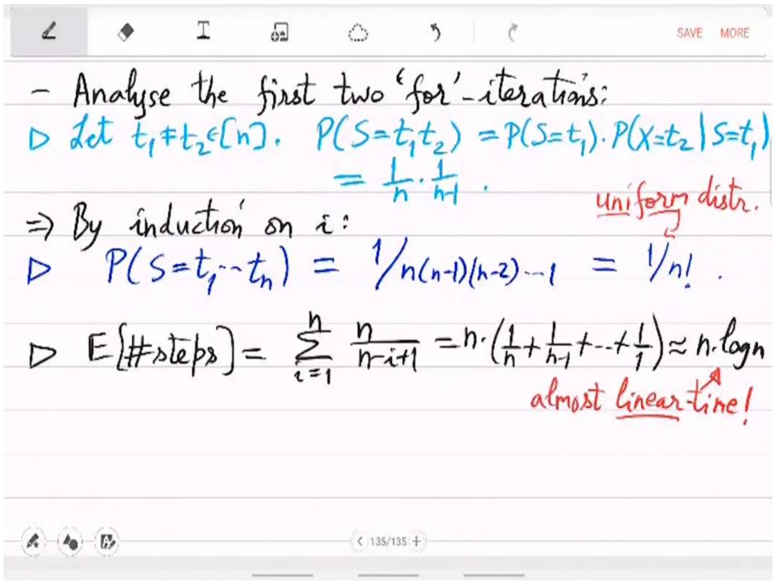
```
Algo:  0) $S \leftarrow$ empty-string;
       1) For $i \in [n]$ {
              2) $X \leftarrow$ rnd number in $[n]$;
              3) If $X$ is in $S$, then GoTo (2);
                                 else $S \leftarrow S \cdot X$   (Append $X$ in the end)
          }
       4) Output $S$;
```

So, keep picking an element X that is new. So, random variable X; X 1 will be something out of 1 to n; then, X 2 will be, you keep trying for X 2 till you get something different from X 1. And then, X 3 you keep trying till you get something different from X 1 and X 2 and so on. So, that obviously gives you a permutation in the end; but what is the chance? Is it uniform distribution? And use this to grow string S by X, by appending X in the end.

So, let us see the algorithm. So, S is just the empty string first initialised. So, when we are looking for the ith element of the string, what we do is of course pick this random number. And if X is in S already, then, this was a useless attempt; so, you have to retry. So, GoTo 2. This seems to be an infinite loop. So, it may happen again and again that you keep picking an element which is already there in S.

And so, it goes on and on. But obviously, this is theoretically infinite; but practically, you are expected to, after a while, get a new element and move forward. Then what will happen is, you will grow the string S. Let us grow it by appending in the end. So, this way, you get all the n elements and you output the string S. So, all that is left is to analyse what is the probability of getting a fixed permutation or favourite permutation. What is the chance of getting that in this algorithm? So, as we did before, analyse the first two iterations.

(Refer Slide Time: 16:44)



- Analyse the first two 'for'-iterations:

▷ Let $t_1 \neq t_2 \in [n]$. $P(S=t_1t_2) = P(S=t_1)\cdot P(X=t_2 \mid S=t_1)$

$= \frac{1}{n}\cdot\frac{1}{n-1}$.

uniform distr.

⇒ By induction on i:

▷ $P(S=t_1\cdots t_n) = 1/n(n-1)(n-2)\cdots 1 = 1/n!$.

▷ $E[\#steps] = \sum_{i=1}^{n} \frac{n}{n-i+1} = n\cdot\left(\frac{1}{n}+\frac{1}{n-1}+\cdots+\frac{1}{1}\right) \approx n\cdot \log n$

almost linear time!

So, what you will get is, for 2 different elements t 1, t 2, the probability that you get the string t 1 followed by t 2. This is that probability originally S was t 1; then, given that previous S is t 1, you pick X to 2. So, what is this? Picking S to be t 1 is 1 by n probability. And based on that, picking t 2 will be 1 over m - 1. And now, you can induct on this. So, by induction on i, what you will get is, probability that the string is this permutation, these elements t 1 to t n, distinct elements in some order; probability of getting this is 1 over n factorial.

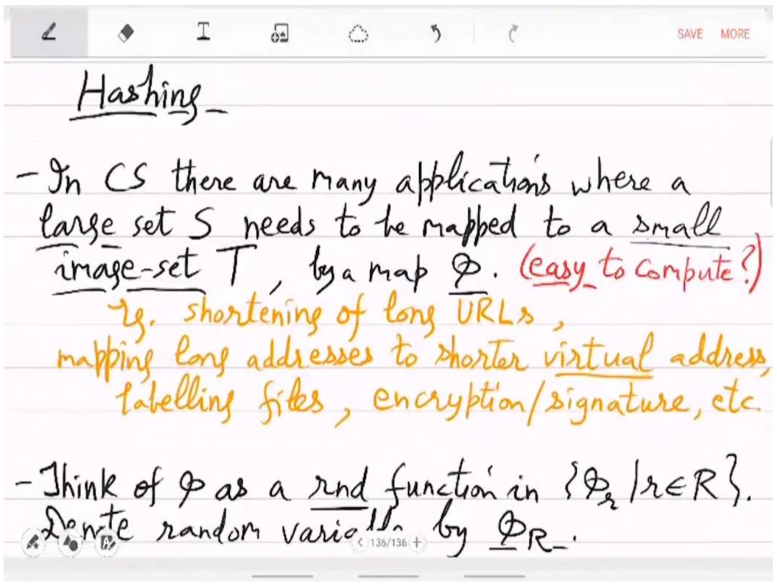
You basically get n; then n - 1; then n - 2 and so on; which is 1 over n factorial. So, that is great. So, there are n factorial permutations, and getting any one of them is equally likely. So, that is the uniform distribution for you. What is the expected number of steps? Because, remember, the algorithm is theoretically an infinite loop; so, we have to see, we have to analyse or give some guarantees on what happens in practice.

So, all we can do is, we can calculate the expectation. So, the expectation for the ith step, the infinite loop in the ith step, it is looking for an X that is new; and i - 1 has been already picked. So, this is like before; following expectation of a geometric distribution, you get this 1 over probability. So, that is n - i + 1 over n, and take the reciprocal; so, which is n times 1 over n; and finally, 1 over 1.

There is a harmonic series; so, you get n log n. It is like integral of dx by X; so, you get log of n. So, you see that, although theoretically it is an infinite loop, practically it is almost a linear time algorithm. It is a very fast algorithm. Its complexity is like sorting n elements, which we considered very fast. So, that completes this sequence of algorithms which are very useful and very fundamental probabilistic algorithms that help you sample elements.

And once you have seen them, you can come up with other kinds of sampling using these as a foundation. So, in a similar spirit, we will now discuss an object that is extremely important in computer science, and probably not studied in math courses.

(Refer Slide Time: 02:19)



Hashing_

- In CS there are many applications where a large set S needs to be mapped to a small image-set T, by a map ϕ. (easy to compute?)
  eg. shortening of long URLs, mapping long addresses to shorter virtual address, labelling files, encryption/signature, etc.
- Think of ϕ as a rnd function in $\{\phi_r \mid r \in R\}$. Denote random variable by Φ_R.

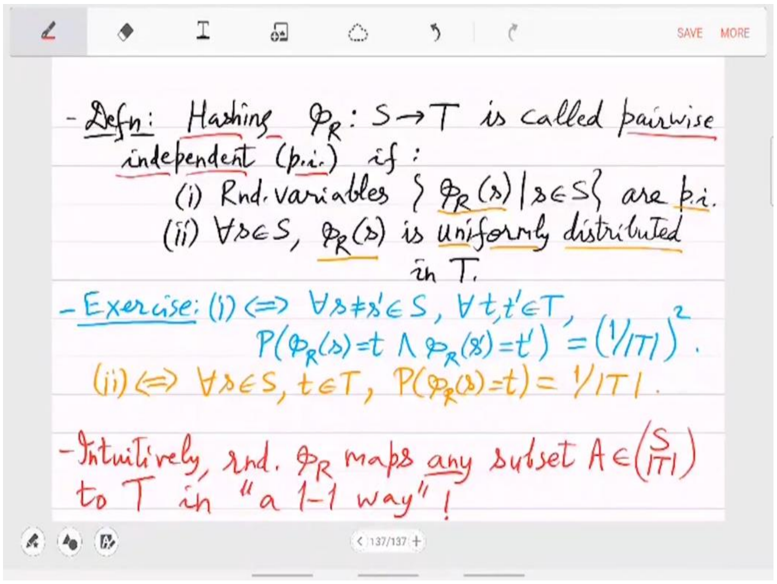
It is called hashing. This is used in almost all areas of computer science, but especially operating systems, databases, of course, theoretical computer science and cryptography. So, basic question here is, in computer science there are many applications where a large set S needs to be mapped to a small set; let us call it image-set T. You want to map S to T, large to small, by a map phi. Why?

Well, you can readily think of these examples, like you have a list of very long website names; so, instead of storing those long website names, you may want to shorten them. So, the simplest lookup table would be, you just use the serial number. So, first website number, second website, third; so, you use 1, 2, 3. Or you can come up with a more elaborate mechanism, because you do not want to always look at the lookup table; you want a function, you want a map, which should be easy to compute.

So, to avoid looking up in a lookup table, you actually want a function that is easy to compute, this function phi. So, like shortening of long website names. So, let us just say URL. And similarly, mapping long addresses in the hard disk or in RAM. There are these long addresses; you want to shorten them in the virtual memory space. Or similarly, you have long file names; so, labelling files.

Or maybe in cryptography; so, let us say encryption or signature. So, these are the few examples that come to mind where computer science needs hashing functions or hash functions. So, we will think of it always as a family of functions that we are after. So, think of phi as a random function in a set phi r. So, you have a family of functions; hopefully each of them is easy to compute, indexed by small r which you should think of as a random string, in the set or in the space big R. So, denote random variable by phi big R. So, that is the random variable which refers to some element of this set phi sub R.

(Refer Slide Time: 26:26)



- Defn: Hashing $\varphi_R : S \to T$ is called pairwise independent (p.i.) if :
  (i) Rnd. variables $\{\varphi_R(s) \mid s \in S\}$ are p.i.
  (ii) $\forall s \in S$, $\varphi_R(s)$ is uniformly distributed in T.
- Exercise: (i) $\Leftrightarrow \forall s \neq s' \in S, \forall t, t' \in T,$
  $P(\varphi_R(s) = t \wedge \varphi_R(s') = t') = (1/|T|)^2$.
  (ii) $\Leftrightarrow \forall s \in S, t \in T, P(\varphi_R(s) = t) = 1/|T|$.
- Intuitively, rnd. φ_R maps any subset $A \in \binom{S}{|T|}$ to T in "a 1-1 way"!

So, now, what do we want in hashing? So, hashing from messages S to image T or from domain S to range T is called pairwise independent; we will shorten it to p.i. So, we call it pairwise independent hash function or hashing family, if random variables phi R s for s in the domain. So, every element in the domain, when you look at its images under phi R, where R is this random variable; it depends on which string you pick, which hash function you pick.

So, phi R s is a random variable. This is a set of random variables. This is pairwise independent. And two is that, for every s, you want phi R s. So, as you change this small r, the image of small s will change, right? Different functions, maybe the image changes. The weight changes, that should be uniform. So, phi R s is uniformly distributed. So, you want phi R s to be pairwise independent, and you want phi R s to be uniformly distributed.

Then we call that it is a good hash function family or it is a good hashing, from domain S to range T. That these hash functions, when you look at some domain element S, the possibilities where it can go from using the hash functions, that is uniform. And when you look at S and S prime, 2 domain elements, then they are pairwise independent, no matter which phi R you pick.

So, some basic properties which are very important; I will leave it as an exercise. This property 1 that we are saying, it basically means that for different S and S prime in the domain, and for all t, t prime in the image, the probability that phi R s is T and phi R s prime is t prime; if you pick your 2 favourite domain elements and you randomly pick a hash function from this family, what is the chance that S goes to t and S prime goes to t prime?

So, by pairwise independent, it should be uniform; so, which is 1 by T times 1 by T; so, square of that. Property two, this says that essentially, chance of S going to t is 1 by T. So, for all s in S and t in T, the probability that phi R s is t is 1 over size of T. So, that is what you want from the random choices of the hash function phi from your family, you want this. And then you call it a good hashing, which is, formally we are calling it p.i. hashing, pairwise independent hash function family.

Just to give the intuition; so, intuitively, random phi R maps any subset A, which has size not more than T. So, let us take this, a subset of S domain elements, not more than the size of T, the range. If you look at the action of phi R on A, it is 1 to 1, in a 1 to 1 way. So, that is what is happening intuitively. This is a 1 to 1 action, but obviously, there might be some bad phi R's, but if you see this probabilistically, then most of the phi R's are good. They are actually, for any subset A of size T, they are just changing the labels in a isomorphic way.