Randomized Algorithms : a very short introduction

Randomized algorithm are algorithms using random bits! When ordinary deterministic algorithms look at its input to determine what actions to take, a randomized algorithm sometimes tosses a coin when making a decision. A consequence is that the randomized algorithm will behave differently every time it is invoked while the deterministic algorithm makes the same choices every time. Instead of having worst cases of input, randomized algorithms will a possibility of bad luck for every input. There is always a chance that decisions made based on random bits are bad and lead to long running times, bad approximations or just nowhere. The key to using randomized algorithms is to make this probability of a bad luck sufficiently small.

The randomness (i.e. random numbers) can be used in a variety of ways. However there are ways of classifying the algorithms by how they report their result. We can classify randomized algorithms in the following groups

Numerical algorithm A sort of randomized approximation algorithm. A numerical algorithm calculates its answer with a certain precision and a certain probability. The result is typically an interval of confidens. Suppose an algorithm for Monte Carlo integration is used to calculate the intergral I of some expression over some domain. The program give us the following result.

$$\mathbb{P}(1.3245 \leq I \leq 1.6533) = 0.99$$

This means the with probability 0.99 the integral I is between 1.3245 and 1.6533. So even if we have a result of a certain precision, we cannot be sure that this interval covers the correct answer, but it does so with probability 0.99. The result resembles the result of a statistical investigation, we have a parameter of interest, a confidence interval and a level of confidence.

Monte Carlo Algorithm When using a Monte carlo algorithm for a problem we will always get an answer, but not always a correct one. If the algorithm is p-correct it gives a correct answer with probability p. Typically the randomness used in a Monte carlo algorithm is used to guide the algorithm to its answer in a faster way with risk of making errors and not beeing able to detect these errors. We cannot determine if a single answer given by a Monte Carlo algorithm is correct, but we can run the algorithm several times and compare the results.

An example of a Monte Carlo algorithm is a simple algorithm for testing if a given integer is a prime or not. Given an integer n the idea is to find $a \in \{2, ..., n/2\}$ such that n is divisible by a. If there are no such a then n is a prime number. A randomized version is to sample 10 numbers between 2 and $n/2$ and if any of these is a divisor of n we answer false, that is, that n is not a prime, and otherwise true.

Las Vegas algorithms A Las Vegas algorithm always gives a correct answer, but the execution time is random. Las Vegas algorithms could be algorithms where randomness is used to find a shorter route to a correct answer. Sometimes this strategy fails and an error occur when the chosen route chosen doesn't end in a solution. If that happen the algorithm start all over again until it has a valid solution.

An example on a Las vegas algorithms is quicksort with a randomly chosen pivot-element. The basic idea in quicksort (qs) is the following. Given an array A, quicksort choose a so called pivot-element $p \in A$ and divides the array in two parts, one

$$A_1 = \{q \in A : q < p\}$$

containing elements smaller than p and the other

$$A_2 = \{q \in A : q > p\}$$

containing elements larger than p. It then recursively sort $A_1,\ A_2$ and return

$$qs(A_1) \cup \{p\} \cup qs(A_2)$$

No matter how the pivot-element is chosen the algorithm will give a correct answer, but a bad choice will result in a long running time. A randomized version choose the pivot-element at random in A, and has excepted running time on $\mathcal{O}(n \log n)$ on any (even previously sorted arrays) input with n elements.

Simulation algorithms Any algorithm used to simulate the behaviour of some phenomenon in our world. We could simulate queuing behaviour at the main entrance to Liseberg during July or performace of a large network, like the Internet. Whenever a system is too complex to treat explicitly via mathematical formulas we can use simulations to learn something about the behaviour of a system. Simulation are sometimes used as a component in the other types of algorithms mentioned above.

The analysis of a randomized algorithm can be done in many different ways. We could analyse running time and get an upper bound on the average running time or we could analyse the required amount of data storage needed during execution, and so on. The purpose of the analysis is the same as it is for deterministic algorithms, to get bounds on the performance. Often the analysis is more involved than for deterministic algorithms since there is an extra component to consider, the randomness. If we consider the last category, the simulation algorithms, we can do the same analysis of running time and/or memory requirements for all of them, and we will. But instead of analysing every algorithm one by one we will use a general approach and analyse a whole class of simulation algorithms at the same time. The main topic of this course is to analyse this class of simulation algorithms, and to do this we need some mathematics. But first, let us hae a look at a small simulation example, and see what mathematical components we need.

A small example

Consider a small queuing system at a gateway on the Internet. Packages are coming in at a certain rate (packages per second), they are handled, and sent away at a certain (not necissarily the same) rate.

No matter how we simulate this system we need to keep track of where packeages are at each time, and how long it takes for the system to handle them. When we look at the system and ask what *state* it is in the answer is perhaps $\{45, 67, 2, 34\}$ meaning that there are 45 packages waiting in Q_1, 67 in Q_2, and so on. The state changes every time a package arrive to Q_1, leave Q_2, Q_3 or Q_4, or in just moved within the system. How packages are transferred into, within and out of the system depends are characteristics we know in advance. For example if a packages arrives with an email header the system

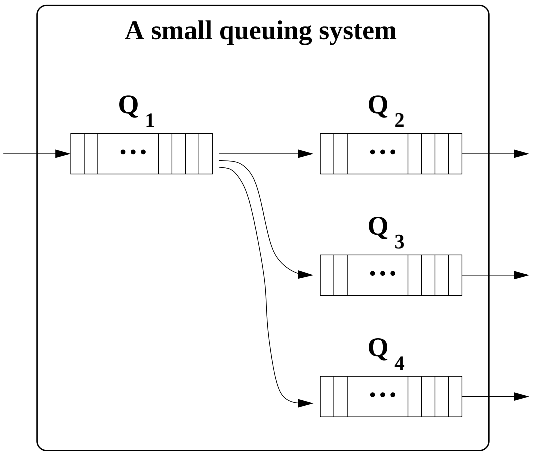

Figure 1: A schematic picture of a gateway on the internet.

know what to do with it, though the time it takes to perform the task is random (lookup adresses and/or translating between protocols).

Given the state at time t_n we know how to randomly update the system to time t_{n+1}, that is, if there are packages wating in Q_1 we know the probability that such a package is moved to Q_2, Q_3 or Q_4. If there are packages waiting in some other queue we know what to do with them also. So a simulation algorithm for this system could be a subroutine taking the state $X_{t_n} = \{q_1, q_2, q_3, q_4\}$, update it according to the system, and return state $X_{t_{n+1}}$. An important property of such a simulation algorithm is that the future of the system depends on the past *only through the present* state of the system. How it got to the present state is irrelevant.

Now lets formulate this in mathematical terms.

Sample space We let Ω be the sample space, that is the set of all possible states of the system.

State Let X_t be the state of the system at time t. Then $\{X_0, X_1, ...\}$ describes what happen with the system for all times.

Update function Let $f : \Omega \to \Omega$ be the update function of the system. Given a state $i \in \Omega$ it tosses a coin and use probabilities $\mathbb{P}(X_{n+1} = j|X_n = i)$ to return the next state $j \in \Omega$.

Given this we have a sequence of random variables $\{X_n\}_{n=1}^{\infty}$ to analyse. Now ...

What questions are reasonable to ask ?

As in any simulation situation we get what we give! The better we tune the system towards a real life system the better the results are. Input in this case are the distributions we use to create packages and to determine handling times. This means that output comes as distributions, that is, we can calculate the probability, $\mathbb{P}(X_n = i)$, that the sequence is in a certain state at time n. Distributions in, distributions out! Any question regarding distributions such as expected values, variances, probabilities for certain values, and so on, are reasonable to ask.

A peek at Markov chains

From basic courses in probability and statistics you are familiar with the concept of random variables and sequences of such. A typical situation is . . .

> *Given random variables $X_1, ..., X_n$ which all have the same distribution and all are independent estimate the expected value and variance.*

To analyse sequence of random variables originating from simulations like the one described above we cannot assume independence, since the value of X_{n+1} depend on the value of X_n. We have to use another model instead, one that allow certain dependencies, and the one we use is Markov Chains. The main feature of Markov chains is the so called Markov property, or the memoryless property, and we can express it in words as . . .

The future depend of the past only through the present.

. . . or in mathematical terms . . .

$$\mathbb{P}(X_{n+1} = i_{n+1} | X_0 = i_0, X_1 = i_1, ..., X_{n-1} = i_{n-1}, X_n = i_n) = \mathbb{P}(X_{n+1} = i_{n+1} | X_n = i_n)$$

The exact mathematical definition is more involved but the general idea is the same.

Why do we use this model? Simulation algorithms evolves in steps, and the set of possible states is large but finite since the available memory is finite. Given the simulation algorithm and the probability distribution of the first element X_0 we can make precise statements about the future and the distributions of X_n for $n \geq 1$.