Support - set of outcomes with positive probability

Bernoulli Distribution: $[0, 1]$ with $[1 - p, p]$
E[Bernoulli] $= p$
Var[Bernoulli] $= p(1 - p)$
$P_X(s) = ps + q$

Binomial Distribution (n, p): $[0, 1, \dots, n]$ with $[\binom{n}{0}p^0(1-p)^n, \dots]$
E[Binomial(n, p)] $= np$
Var[Binomial(n, p)] $= np(1 - p)$
$P_X(s) = (ps + q)^n$

Geometric Distribution: $[0, 1, 2, \dots]$ with $[p, p(1-p), p(1-p)^2, \dots]$
E[Geometric] $= \frac{1-p}{p}$
Var[Geometric] $= \frac{1-p}{p^2}$
$P_X(s) = \frac{p}{1-qs}$

Poisson Distribution: $[0, 1, 2, \dots]$ with $[e^{-\lambda}\frac{\lambda^k}{k!}]$
E[Poisson] $= \lambda$
Var[Poisson] $= \lambda$
$P_X(s) = e^{\lambda(s-1)}$

Gambler's Ruin: start with x and $++$ with p and $--$ with $1 - p$ and halt at a or 0.

Generating Functions - special case of Fourier/Laplace harmonic analysis way to capture probability distribution into function

$P_X(s) = \sum_{k=0}^{\infty} p_k s^k$

$P_{X+Y}(s) = P_X(s)P_Y(s)$ convolution by definition

$Y = \sum_{k=1}^{N} g_k$ on N independent distribution then $P_Y(s) = P_N(P_g(s))$

$\mathrm{E}[X], \mathrm{E}[X(X-1)], \mathrm{E}[X(X-1)(X-2)], \dots$ factorial moments given by kth derivative of $P_X(s)$ evaluated at $s = 1$ whence we may deduce moments $\mathrm{E}[X^k]$ linearly. And note $\mathrm{Var}[X] = P''(1) + P'(1) - (P'(1))^2$

$\mathrm{E}[X] = \mathrm{E}[X]$
$\mathrm{E}[X^2] = \mathrm{E}[X(X-1)] + \mathrm{E}[X]$
$\mathrm{E}[X^3] = \mathrm{E}[X(X-1)(X-2)] + 3\mathrm{E}[X(X-1)] + \mathrm{E}[X]$

Branching process has offspring distribution p_n then if $\mathrm{E}[p_n] = u$ with finite variance then by definition $\mathrm{E}[Z_n] = u^n$ and $\mathrm{Var}[Z_n] = \mathrm{Var}[p_n]u^n\frac{1-u^{n+1}}{1-u}$ if $u \neq 1$ and $\mathrm{Var}[p_n](n+1)$ if $u = 1$

Extinction probability p_E is smallest non-negative solution of the equation $x = P(x)$ where P is the generating function of the offspring distribution.

$i \to j$, i communicates with j, j is a consequent of/accesible from/follows i - nonzero P of arriving at j from i

$i \leftrightarrow j$, i and j intercommunicate if $i \to j$ and $j \to i$ a set B of states is irreducible if so classes of a chain, maximal irreducible sets, are strongly connected components in this digraph.

A set of states is closed if once there never escapes i.e. sink component.

A closed singleton is an absorbing state.

Recurrent if return with P = 1, positive recurrent if the Expected Value of time between 2 visits is finite, null if the EV is infinite, transient if positive P of not returning.

Recurrence and transience are class properties i.e. belong to all members of the same connected component.

Class on a finte state space is recurrent if and only if it is closed.

Write transition matrix P in the canonical form:
$\begin{bmatrix} PC & 0 \\ R & Q \end{bmatrix}$

So Q is T to T edges and PC is C to C edges and R is T to C edges where T is transient states i.e. union of all transient classes and PC is recurrent/closed states i.e. union of all sink connected components whence:

$U = (I - Q)^{-1}R = (1 + Q + Q^2 + \dots)R = FR$ is transition probabilities

F captures expected number of times hitting each state in T prior to transitioning into C

To produce the expected hitting times from transient states to recurrent states one can solve a system of linear equations in terms of the expected hitting times like $(1 - Q)x = 1$ using Stephen Wolfram Mathemaica, MatLab, Eigen [does not support 128 bit floating point computations well], or maybe not SciPy Sparse in sparse settings.

This comes in to reading up a tonne on interesting statistics research. Say Leonhard Euler himself gave a task on computing the expected hitting times for a random walk on the discrete torus $(Z/12Z)^{12}$ until hitting the state of a diagonal entry. Well maybe this isomorphs in to a sparse matrix with 1352066 rows where each row might have say at most 25 nonzero entries. Read up on linsolve, cgs, pcg, ilu, incomplete LU lower-upper factorisation.

Say instead one reduces the number of rows to 112719 and wanted to solve an explicit dense matrix instance with 128 bit floating point computations. Then perhaps executing in Stephen Wolfram Mathematica on a 2TB Windows Professional machine is not a bad idea if the Apple Macintosh OS Operating System is handling going beyond 40GB poorly. Or an AWS Amazon Web Services instance.

OK using Mathematica after generating the relevant .mtx and .txt files in C++:

```
Mathematica

lol = Import["b12-12.mtx", "Table"];
dims = lol[[1]];
entries = lol[[2 ;;]];
b = SparseArray[({#[[1]], #[[2]]} -> #[[3]]) & /@ entries,
dims[[1 ;; 2]]];
bb = Import["bb12-12.txt", "Table"];
c = Import["c12-12.txt", "Table"];
x = LinearSolve[N[b, 50], N[bb, 50],
Method -> {"Krylov", "Method" -> "BiCGSTAB",
"Preconditioner" -> "ILU0"}];
answer += (c . x)*2*12/12^12;
```

And read up on all of the papers behind all of the techniques listed on the Stephen Wolfram Mathematica documentation page for LinearSolve.