MORE INDEPENDENT EVENTS

DISCRETE STRUCTURES II

DARRYL HILL

BASED ON THE TEXTBOOK:

DISCRETE STRUCTURES FOR COMPUTER SCIENCE: COUNTING, RECURSION, AND PROBABILITY

BY MICHIEL SMID

Events

English /Logic	Subsets of S
$A \wedge B$	$A \cap B$
$A \vee B$	$A \cup B$
$\neg A$	$\overline{A}$
$A \rightarrow B$	$A \subseteq B$

Often events are described in English, but they are really sets and the English is describing a logical predicate that selects a subset of the sample space. Consider $S = \text{all people}$

$$A = \text{Everyone with black hair and brown eyes}$$
$$B = \text{Everyone with black hair or brown eyes}$$

$$C = \text{Everyone with black hair}$$
$$D = \text{Everyone with brown eyes}$$

$$A = C \cup D$$
$$A = \{x \mid x \text{ has black hair} \vee x \text{ has brown eyes}\}$$

$$B = C \cap D$$
$$B = \{x \mid x \text{ has black hair} \wedge x \text{ has brown eyes}\}$$

Events

English /Logic	Subsets of S
$A \wedge B$	$A \cap B$
$A \vee B$	$A \cup B$
$\neg A$	$\overline{A}$
$A \rightarrow B$	$A \subseteq B$

Often events are described in English, but they are really sets and the English is describing a logical predicate that selects a subset of the sample space. Consider $S = \text{all people}$

$$A = \text{People who does not have black hair}$$
$$B = \text{People with blond hair}$$

$$C = \text{People with black hair}$$

$$A = \overline{C} \quad \text{(set operation)}$$
$$A(x) = \neg C(x) \quad \text{(logically equivalent)}$$

$$B(x) \rightarrow A(x)$$
$$B \subseteq A$$

Circuit C: There are n components

$$C_1, C_2, \ldots, C_n$$

$$\forall i, 1 \le i \le n: \quad A_i = \mathrm{C_i} \text{ fails}$$

$$\Pr(A_i) = p, \qquad 0 < p < 1$$

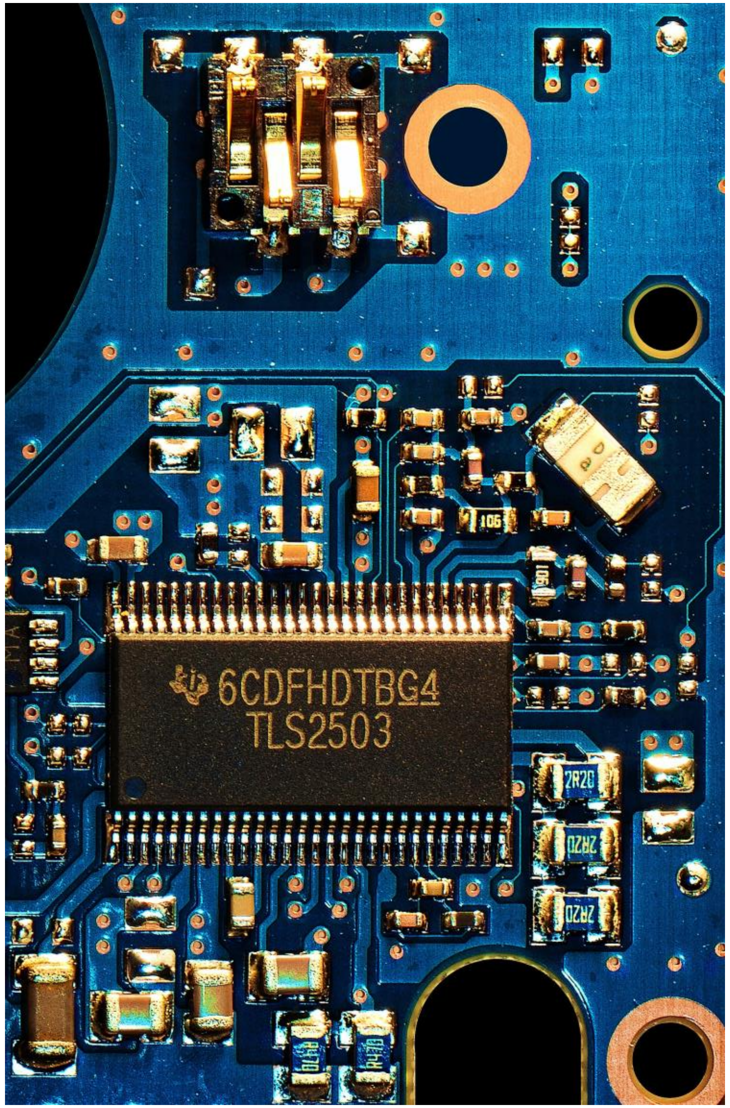
$A_1, \ldots, A_n$ are mutually independent

1. C fails if ≥ 1 component fails.

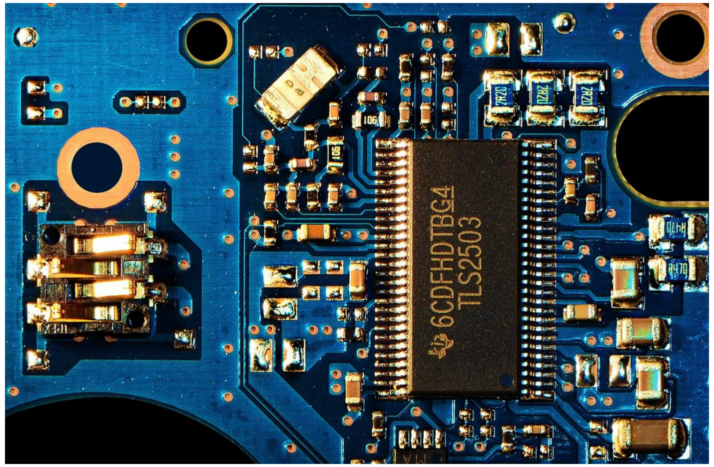
$A = C$ fails.

What is $\Pr(A)$?

We can write the event A in terms of $A_1, A_2, \ldots, A_n$



Circuit C: There are n components $C_1, C_2, \ldots, C_n$
$\forall i, 1 \leq i \leq n$: $A_i = \mathrm{C_i}$ fails
$\Pr(A_i) = p,\ 0 < p < 1$
$A_1, \ldots, A_n$ are mutually independent
1. C fails if ≥ 1 component fails.
Event $A = C$ fails.



We can write the event A in terms of $A_1, A_2, \ldots, A_n$:

$$A \leftrightarrow A_1 \vee A_2 \vee A_3 \vee \cdots \vee A_n$$

Since "logical or" corresponds to union, you may think to apply the sum rule. The problem is that these events are not disjoint. That is, if A_1 happens, it may still be that A_2 also happens.

The actual sample space is all possible combinations of circuits failing. The event A_i consists of all outcomes in which circuit C_i fails.

The sample space is all subsets of circuits that fail.

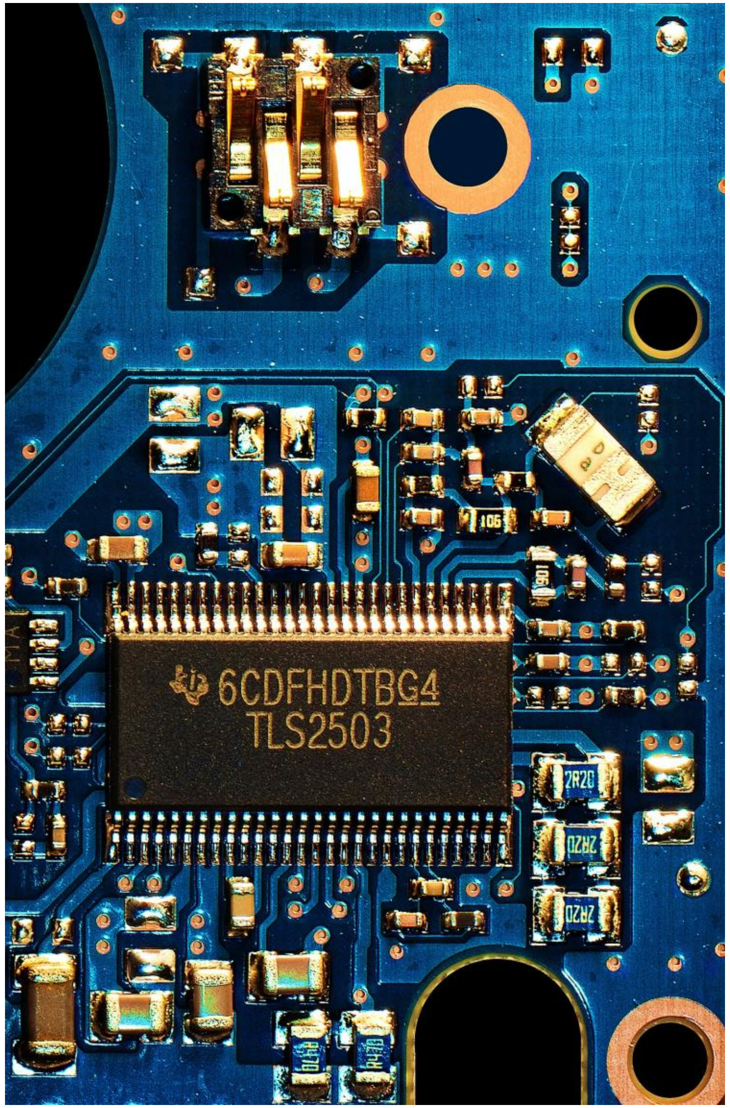
The probability that C_i fails is p.

The probability that C_i does not fail is $(1-p)$.

If there are i circuits that fail, then there are $n-i$ circuits that don't fail.

Then the probability of an outcome in which i circuits fail is:

$$p^i(1-p)^{n-i}$$

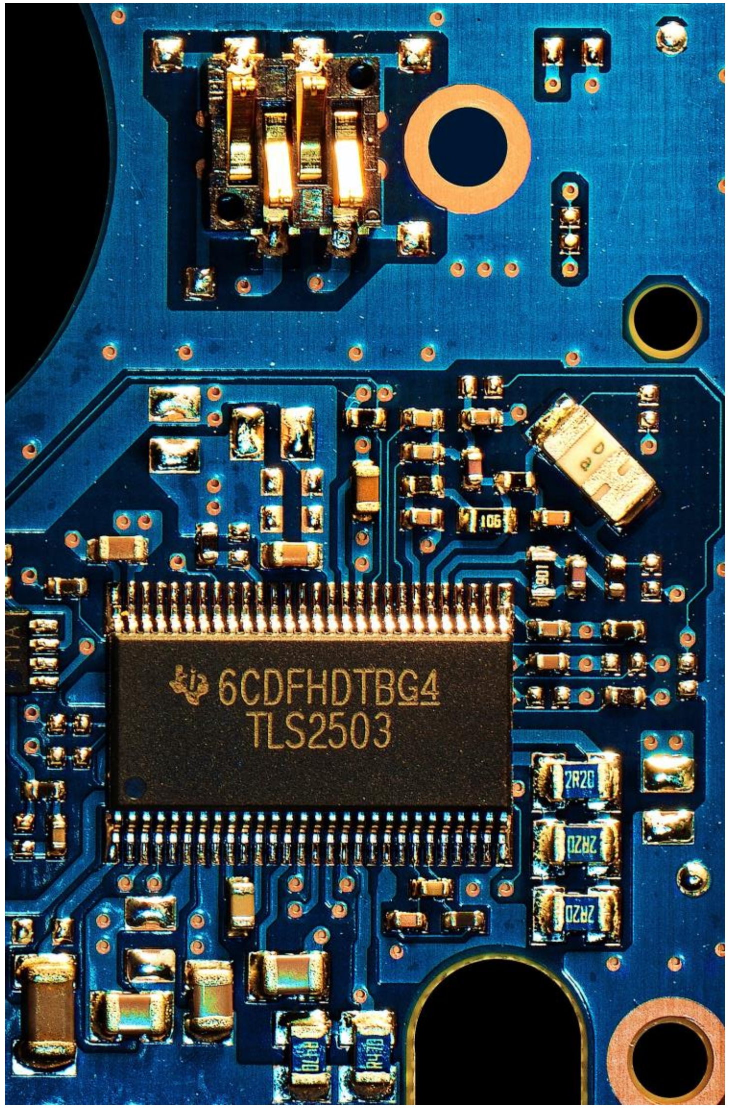


Using Newton, we can verify that all outcomes sum to 1:

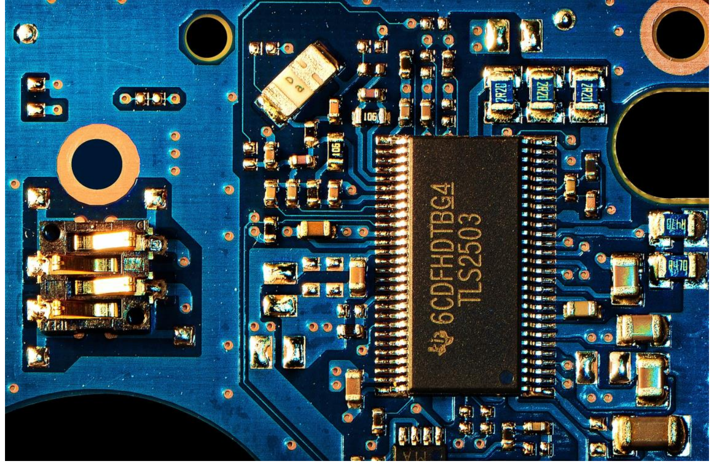
$$(p - (1 - p)^n = \sum_{i=0}^{n} \binom{n}{i} \cdot p^i \cdot (1 - p)^{n-i}$$
$$= 1$$

An event A_i is the sum of the probabilities of all outcomes in which C_i fails. There are $n - 1$ circuits that are not C_i, so to calculate the probability of all outcomes in which C_i fails:

$$\sum_{i=0}^{n-1} \binom{n-1}{i} \cdot p \cdot p^i \cdot (1 - p)^{n-i-1}$$
$$= p \cdot (p - (1 - p)^{n-1}$$
$$= p \cdot 1^{n-1}$$
$$= p$$



Circuit C: There are n components $C_1, C_2, \ldots, C_n$
$\forall i, 1 \leq i \leq n$: $A_i = \mathrm{C_i}$ fails
$\Pr(A_i) = p,\ 0 < p < 1$
$A_1, \ldots, A_n$ are mutually independent
1. C fails if ≥ 1 component fails.
$A = C$ fails.



We can write the event A in terms of $A_1, A_2, \ldots, A_n$:

$$A \leftrightarrow A_1 \vee A_2 \vee A_3 \vee \cdots \vee A_n$$

If we were using "logical and" we could use the fact that these are mutually independent to compute the probability. That is:

$$\Pr(A_1 \wedge A_1 \wedge \cdots \wedge A_n) = \Pr(A_1) \cdot \Pr(A_2) \cdot \ldots \cdot \Pr(A_n)\,.$$

So we want to turn "logical or" into "logical and". We can use DeMorgan's law.

Circuit C: There are n components $C_1, C_2, \ldots, C_n$
$\forall i, 1 \leq i \leq n$: $A_i = \mathrm{C_i}$ fails
$\Pr(A_i) = p,\ 0 < p < 1$
$A_1, \ldots, A_n$ are mutually independent
1. C fails if ≥ 1 component fails.
$A = C$ fails.

$$A \leftrightarrow A_1 \vee A_2 \vee A_3 \vee \cdots \vee A_n$$

By DeMorgan's:

$$\bar{A} \leftrightarrow \overline{A_1} \wedge \overline{A_2} \wedge \overline{A_3} \wedge \cdots \wedge \overline{A_n}$$

And

$$\Pr(\bar{A}) = \Pr(\overline{A_1} \wedge \overline{A_2} \wedge \overline{A_3} \wedge \cdots \wedge \overline{A_n})$$

$$= \Pr(\overline{A_1}) \cdot \Pr(\overline{A_2}) \cdot \ldots \cdot \Pr(\overline{A_n})$$

We know these terms.

$$\Pr(\overline{A_i}) = 1 - \Pr(A_i)$$
$$= (1 - p)$$

Thus:

$$\Pr(\bar{A}) = \Pr(\overline{A_1}) \cdot \Pr(\overline{A_2}) \cdot \ldots \cdot \Pr(\overline{A_n})$$
$$= (1 - p) \cdot (1 - p) \cdot \ldots \cdot (1 - p)$$
$$= (1 - p)^n$$
$$\Pr(A) = 1 - \Pr(\bar{A})$$
$$= 1 - (1 - p)^n$$

When $n \to \infty$:

$$\lim_{n \to \infty} [1 - (1 - p)^n]$$
$$= 1 - 0$$
$$= 1$$

Circuit C: There are n components $C_1, C_2, \ldots, C_n$

$\forall i, 1 \leq i \leq n$: $A_i = \mathrm{C_i}$ fails

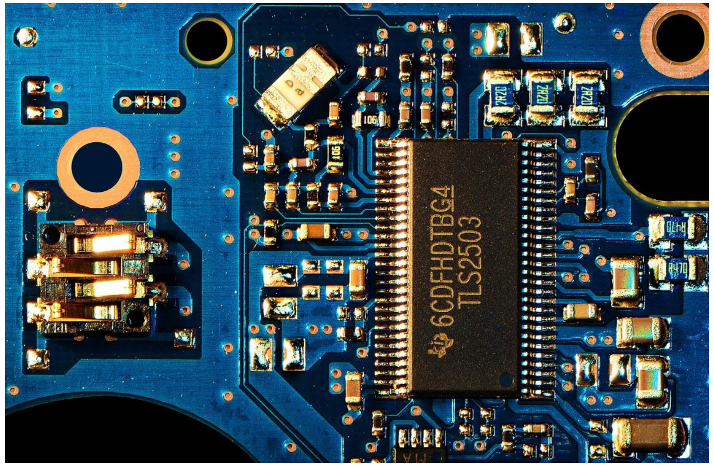
$\Pr(A_i) = p,\ 0 < p < 1$

$A_1, \ldots, A_n$ are mutually independent

1. C fails if ≥ 1 component fails.

$A = C$ fails.

$$\lim_{n\to\infty} \Pr(A) = 1$$



2. C fails when all components fail

$A = C$ fails.

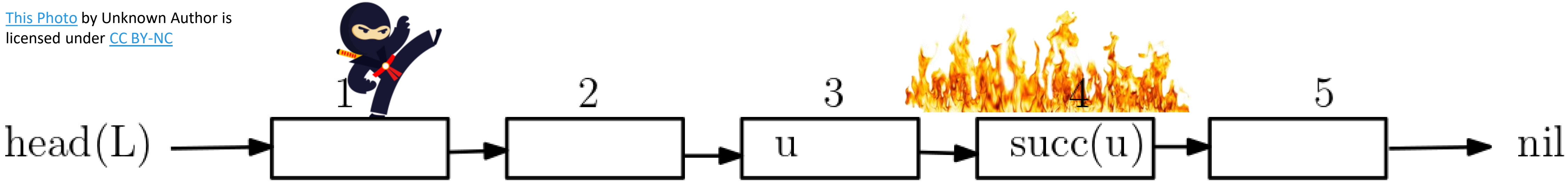
$$A \leftrightarrow A_1 \wedge A_2 \wedge \cdots \wedge A_n$$

$$\Pr(A) = \Pr(A_1 \wedge A_2 \wedge \cdots \wedge A_n)$$
$$= \Pr(A_1) \cdot \Pr(A_2) \cdot \ldots \cdot \Pr(A_n)$$

(because mutually independent)

$$= \overbrace{p \cdot p \cdot \ldots \cdot p}^{\text{n times}}$$
$$= p^n$$

$$\lim_{n\to\infty} \Pr(A)$$
$$= \lim_{n\to\infty} p^n$$
$$= 0$$



Linked List with n nodes

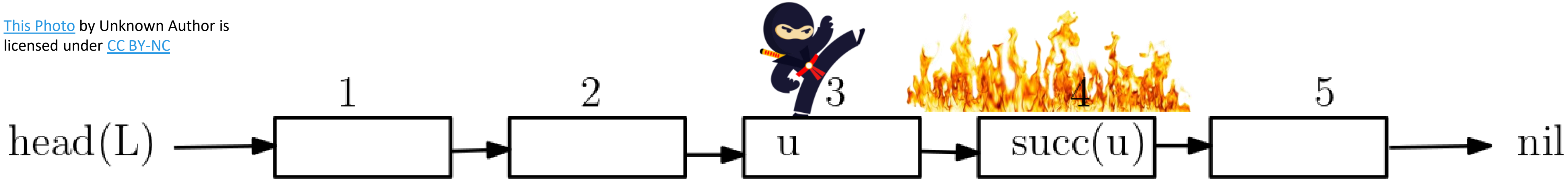
In a linked list we start at head. If we want to see element 3, then we must walk through the linked list from head to 1 to 2 to 3.

Task: Choose a uniformly random node in L.

That means we choose any node $\in \{1..5\}$ with equal probability, i.e., $\frac{1}{5}$, or in general for a list of n elements, $\frac{1}{n}$.

As a tool we have a function $\mathrm{Random(i)}$ which returns an integer from the range $\{1..i\}$ uniformly at random.

The value given by $\mathrm{Random(i)}$ does not depend on previous or future calls to $\mathrm{Random(i)}$ – i.e., they are mutually independent.



Linked List with n nodes

Task: Choose a uniformly random node in L.

$\mathrm{Random}(i)$ returns a uniformly random element in the range $\{1,2,3,\ldots,i\}$, independent of previous calls to Random().

Scenario 1. We know n.

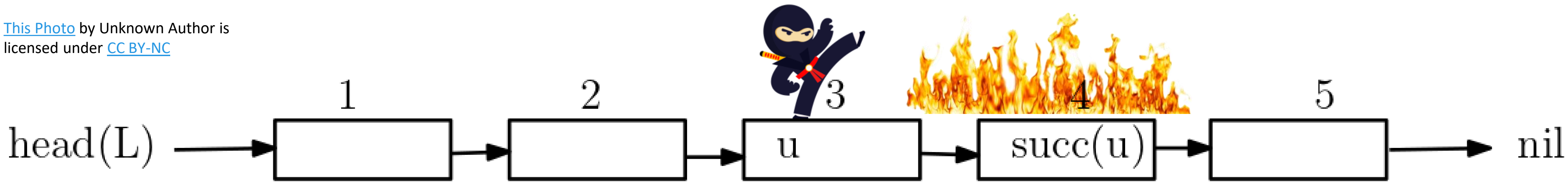
If we know n then there is a simple solution.

Algorithm n is known:

Let $i = \mathrm{Random}(n)$.

Walk through the linked list until we are at location i.

Return the element at location i.



Linked List with n nodes

Task: Choose a uniformly random node in L.

$\text{Random}(i)$ returns a uniformly random element in the range $\{1,2,3,\ldots,i\}$, independent of previous calls to Random().

1. We know n.
2. We don't know n.

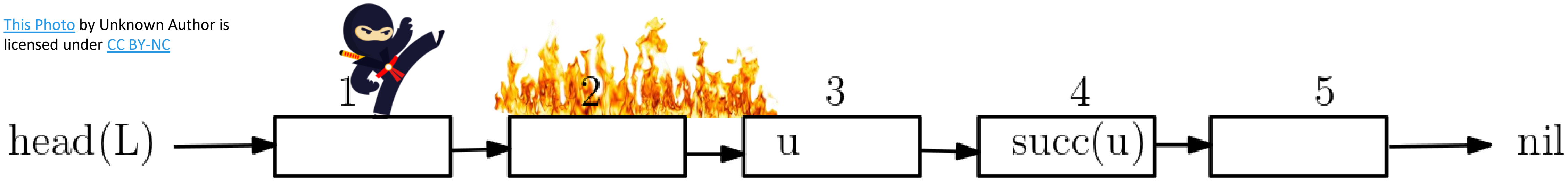
We can find n by traversing the list once and counting all elements.

Algorithm n is not known:
Let $i = \text{Random}(n)$.

Walk through the linked list until we are at location i.

Return the element at location i.

Requires 2 traversals.



Linked List with n nodes

Task: Choose a uniformly random node in L.

$\mathrm{Random}(i)$ returns a uniformly random element in the range $\{1,2,3,\ldots,i\}$, independent of previous calls to Random().

1. We know n.
2. We don't know n.
3. We don't know n, we can traverse only once (it's a bit stream).

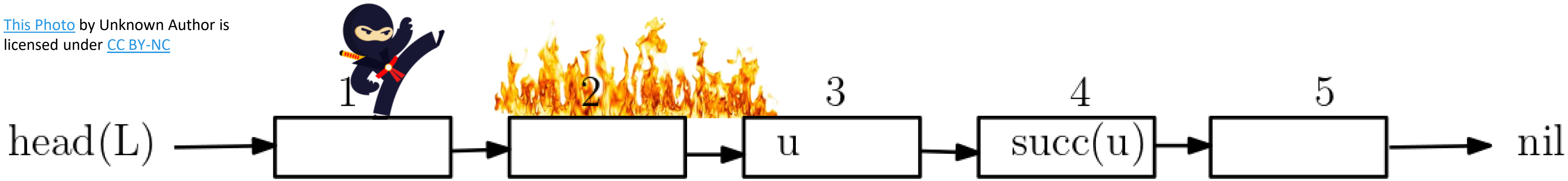
We can do one traversal, front to back –

(the list is on fire and you are being pursued by ninjas)

and we must return a node (say the head) with probability $\frac{1}{5}$.

But we do not know there are 5 nodes.

Here is the algorithm:




```
ReturnRandomNode(L):

u = head(L)
i = 1          // nodes seen so far
while u ≠ nil:
     r = Random(i)
     if r = 1, x = u
     u = succ(u)
     i = i + 1
return x
```

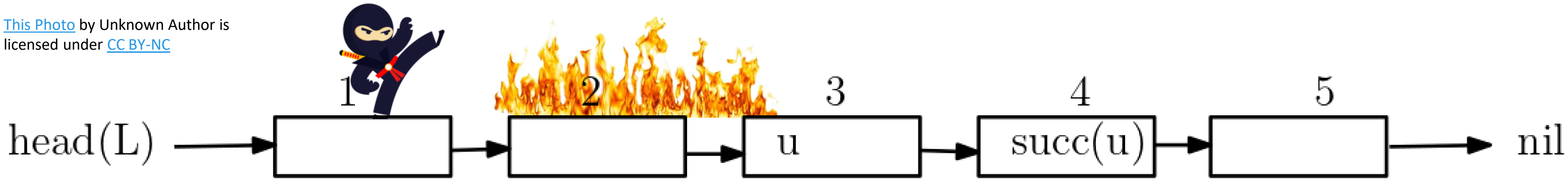
Say we are at element i.

`r = Random(i)`

Thus $r \in \{1..i\}$, and

$$\Pr(r = 1) = \frac{1}{i}$$

If $r = 1$ then we select element u.
So element u is selected (at this time) with probability $\frac{1}{i}$.



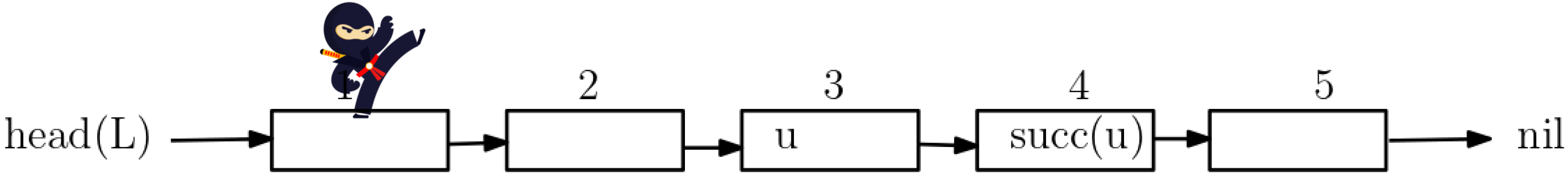

```
ReturnRandomNode(L):

u = head(L)
i = 1          // nodes seen so far
while u ≠ nil:
    r = Random(i)
    if r = 1, x = u
    u = succ(u)
    i = i + 1
return x
```

Starting at location 1, we select the head with probability $\frac{1}{1} = 1$.

Location 2: $r = \mathrm{Random(i)}$, thus $r \in \{1,2\}$.

$\Pr(r = 1) = \frac{1}{2}$.




```
ReturnRandomNode(L):

u = head(L)
i = 1      // nodes seen so far
while u ≠ nil:
     r = Random(i)
     if r = 1, x = u
     u = succ(u)
     i = i + 1
return x
```

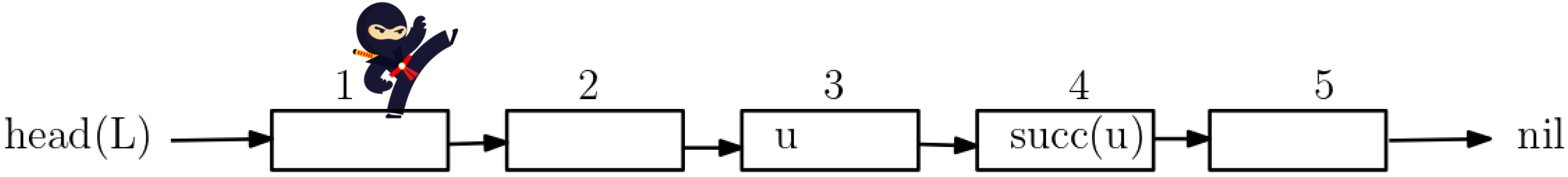
First time through we choose head(L) with probability 1.

Next Loop: i = 2. if r = 1, x = u
if r =2, x = head(L)

Next Loop: i = 3. if r = 1, x = u
if r ={2,3}, x doesn't change

Next Loop: i = 4. if r = 1, x = u
if r ={2,3,4}, x doesn't change

etc.




```
ReturnRandomNode(L):

u = head(L)
i = 1      // nodes seen so far
while u ≠ nil:
    r = Random(i)
    if r = 1, x = u
    u = succ(u)
    i = i + 1
return x
```

First time through we choose head(L) with probability 1.

$\Pr(x = u) = 1$

Next Loop: i = 2. if r = 1, x = u $\quad \Pr(x = u) = \frac{1}{2}$

if r =2, x = head(L)

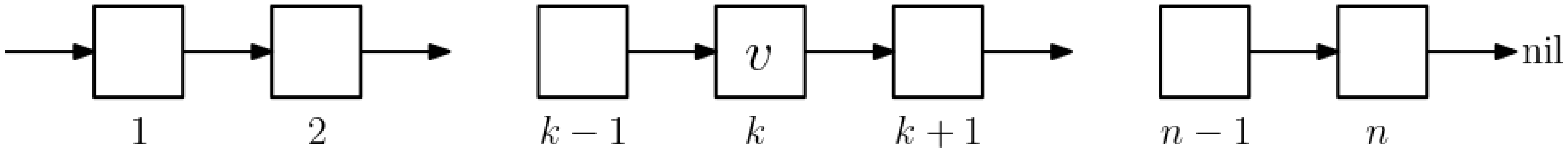
Next Loop: i = 3. if r = 1, x = u $\quad \Pr(x = u) = \frac{1}{3}$

if r ={2,3}, x doesn't change

Next Loop: i = 4. if r = 1, x = u $\quad \Pr(x = u) = \frac{1}{4}$

if r ={2,3,4}, x doesn't change

etc.




```
ReturnRandomNode(L):

u = head(L)
i = 1      // nodes seen so far
while u ≠ nil:
    r = Random(i)
    if r = 1, x = u
    u = succ(u)
    i = i + 1
return x
```

At the end of the algorithm, we know n.

We will select an arbitrary node v at location k.

To show: $\Pr(x = v) = \frac{1}{n}$.

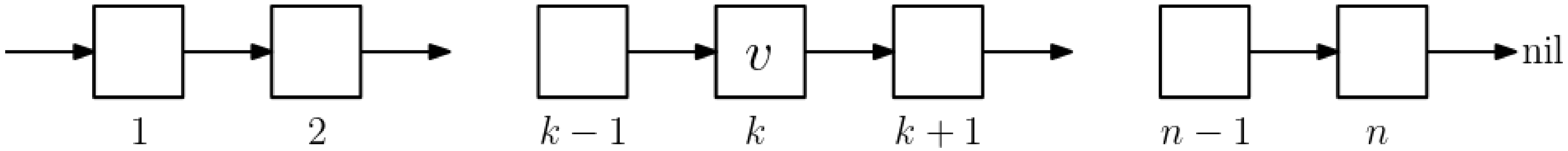
Events:

$$A = \text{Algorithm returns } v$$

For $1 \leq i \leq n$:

$$A_i = x \text{ changes during iteration } i$$

We know $\Pr(A_i) = 1/i$.




```
ReturnRandomNode(L):

u = head(L)
i = 1      // nodes seen so far
while u ≠ nil:
     r = Random(i)
     if r = 1, x = u
     u = succ(u)
     i = i + 1
return x
```

Events:

$$A = \text{Algorithm returns } v$$

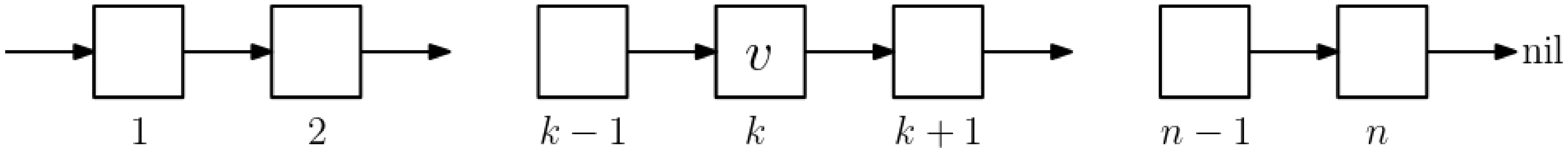
For $1 \leq i \leq n$:

$$A_i = x \text{ changes during iteration } i$$

We know $\Pr(A_i) = 1/i$. Express A in terms of A_i.

If we returned v, then in that step we had $r = 1$ with probability $1/k$.

Does it matter what happened in the previous $k - 1$ steps?




```
ReturnRandomNode(L):

u = head(L)
i = 1      // nodes seen so far
while u ≠ nil:
     r = Random(i)
     if r = 1, x = u
     u = succ(u)
     i = i + 1
return x
```

Events:

$$A = \text{Algorithm returns } v$$

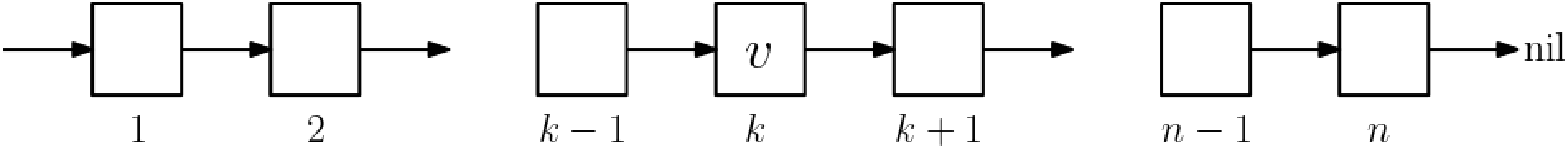
For $1 \leq i \leq n$:

$$A_i = x \text{ changes during iteration } i$$

If we returned v, then in that step we had $r = 1$ with probability $1/k$.

What has to happen now?

In step $k + 1$ it must be that x does not change.
In step $k + 2$ it must be that x does not change.
In step $k + 3$ it must be that x does not change.




```
ReturnRandomNode(L):

u = head(L)
i = 1     // nodes seen so far
while u ≠ nil:
     r = Random(i)
     if r = 1, x = u
     u = succ(u)
     i = i + 1
return x
```

Events:

$A =$ Algorithm returns v

For $1 \leq i \leq n$:

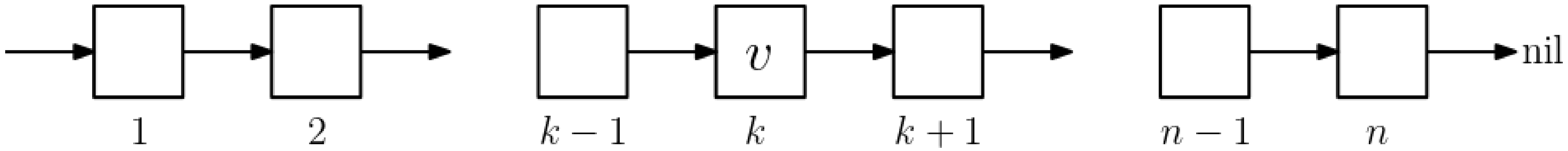
$A_i = x$ changes during iteration i

$$A = A_k \wedge \overline{A}_{k+1} \wedge \overline{A}_{k+2} \wedge \cdots \wedge \overline{A}_n$$

Since the calls to `Random(i)` are mutually independent, and each event A_i is determined by a call to `Random(i)`, all events A_i are mutually independent. Thus:

$$\Pr(A) = \Pr\left(A_k \wedge \overline{A}_{k+1} \wedge \overline{A}_{k+2} \wedge \cdots \wedge \overline{A}_n\right)$$

$$\Pr(A) = \Pr(A_k) \cdot \Pr\left(\overline{A}_{k+1}\right) \cdot \Pr\left(\overline{A}_{k+2}\right) \cdot \ldots \cdot \Pr\left(\overline{A}_n\right)$$




```
ReturnRandomNode(L):

u = head(L)
i = 1     // nodes seen so far
while u ≠ nil:
     r = Random(i)
     if r = 1, x = u
     u = succ(u)
     i = i + 1
return x
```

Events:

$A =$ Algorithm returns v

For $1 \le i \le n$:

$A_i = x$ changes during iteration i

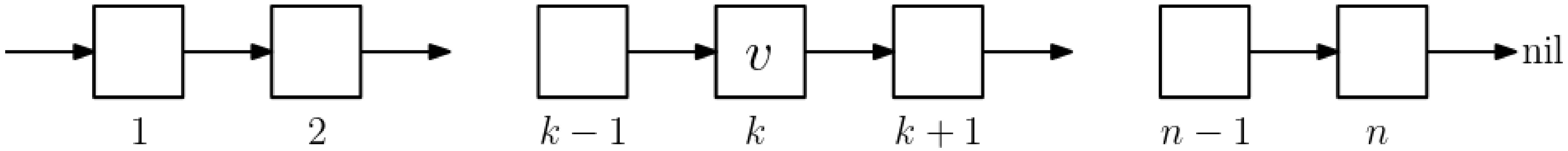
$$\Pr(A) = \Pr\left(A_k \wedge \overline{A}_{k+1} \wedge \overline{A}_{k+2} \wedge \cdots \wedge \overline{A}_n\right)$$

$$\Pr(A) = \Pr(A_k) \cdot \Pr\left(\overline{A}_{k+1}\right) \cdot \Pr\left(\overline{A}_{k+2}\right) \cdot \ldots \cdot \Pr\left(\overline{A}_n\right)$$

$$\Pr(A_k) = \frac{1}{k}$$

$$\Pr\left(\overline{A}_{k+1}\right) = 1 - \Pr(A_{k+1}) = 1 - \frac{1}{\mathrm{k}+1}$$

$$1 - \frac{1}{\mathrm{k}+1} = \frac{k+1}{k+1} - \frac{1}{\mathrm{k}+1} = \frac{k}{k+1}$$




```
ReturnRandomNode(L):

u = head(L)
i = 1     // nodes seen so far
while u ≠ nil:
     r = Random(i)
     if r = 1, x = u
     u = succ(u)
     i = i + 1
return x
```

Events:

$A =$ Algorithm returns v

For $1 \leq i \leq n$:

$A_i = x$ changes during iteration i

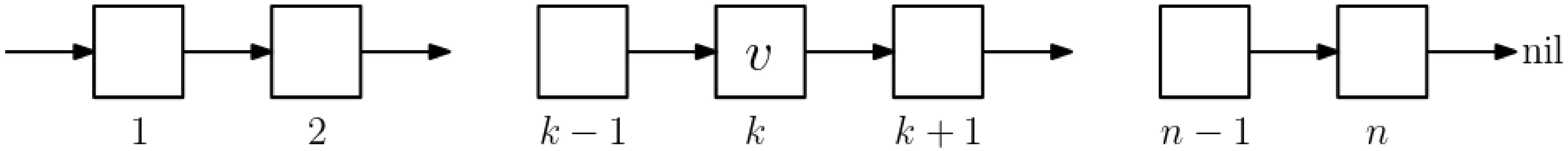
$$\Pr(A) = \Pr\left(A_k \wedge \overline{A}_{k+1} \wedge \overline{A}_{k+2} \wedge \cdots \wedge \overline{A}_n\right)$$

$$\Pr(A) = \Pr(A_k) \cdot \Pr\left(\overline{A}_{k+1}\right) \cdot \Pr\left(\overline{A}_{k+2}\right) \cdot \ldots \cdot \Pr\left(\overline{A}_n\right)$$

$$\Pr(A_k) = \frac{1}{k}$$

$$\Pr\left(\overline{A}_{k+2}\right) = 1 - \Pr(A_{k+2}) = 1 - \frac{1}{\mathrm{k}+2}$$

$$1 - \frac{1}{\mathrm{k}+2} = \frac{k+2}{k+2} - \frac{1}{\mathrm{k}+2} = \frac{k+1}{k+2}$$




```
ReturnRandomNode(L):

u = head(L)
i = 1      // nodes seen so far
while u ≠ nil:
     r = Random(i)
     if r = 1, x = u
     u = succ(u)
     i = i + 1
return x
```

Events:

$A =$ Algorithm returns v

For $1 \leq i \leq n$:

$A_i = x$ changes during iteration i

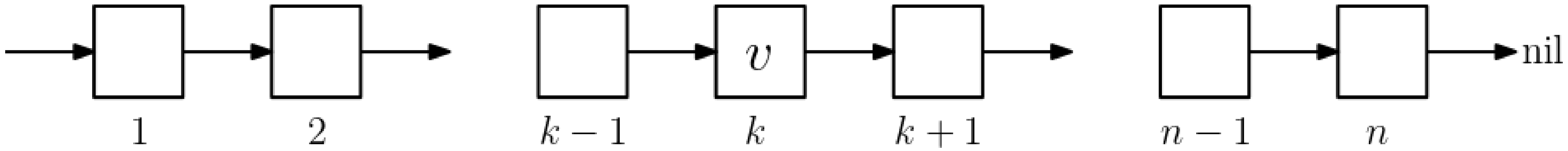
$$\Pr(A) = \Pr\left(A_k \wedge \overline{A}_{k+1} \wedge \overline{A}_{k+2} \wedge \cdots \wedge \overline{A}_n\right)$$

$$\Pr(A) = \Pr(A_k) \cdot \Pr\left(\overline{A}_{k+1}\right) \cdot \Pr\left(\overline{A}_{k+2}\right) \cdot \ldots \cdot \Pr(\overline{A}_n)$$

$$\Pr(A_k) = \frac{1}{k}$$

$$\Pr\left(\overline{A}_{k+3}\right) = 1 - \Pr(A_{k+3}) = 1 - \frac{1}{\mathrm{k}+3}$$

$$1 - \frac{1}{\mathrm{k}+3} = \frac{k+3}{k+3} - \frac{1}{\mathrm{k}+3} = \frac{k+2}{k+3}$$




```
ReturnRandomNode(L):

u = head(L)
i = 1      // nodes seen so far
while u ≠ nil:
     r = Random(i)
     if r = 1, x = u
     u = succ(u)
     i = i + 1
return x
```

Events:

$A = \text{Algorithm returns } v$

For $1 \le i \le n$:

$A_i = x \text{ changes during iteration } i$

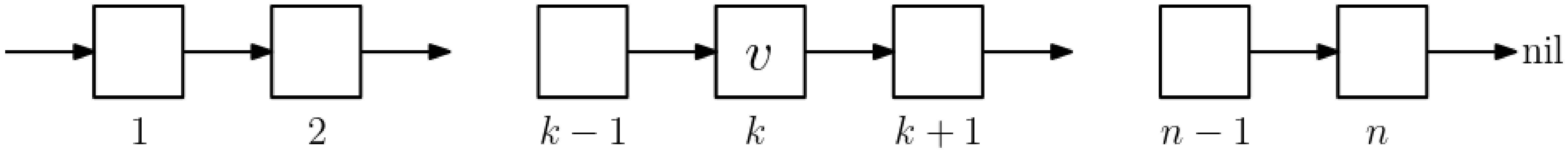
$$\Pr(A) = \Pr\left(A_k \wedge \overline{A}_{k+1} \wedge \overline{A}_{k+2} \wedge \cdots \wedge \overline{A}_n\right)$$

$$\Pr(A) = \Pr(A_k) \cdot \Pr\left(\overline{A}_{k+1}\right) \cdot \Pr\left(\overline{A}_{k+2}\right) \cdot \ldots \cdot \Pr(\overline{A}_n)$$

$$\Pr(A_k) = \frac{1}{k}$$

$$\Pr\left(\overline{A}_n\right) = 1 - \Pr(A_n) = 1 - \frac{1}{\mathrm{n}}$$

$$1 - \frac{1}{\mathrm{n}} = \frac{n}{n} - \frac{1}{\mathrm{n}} = \frac{n-1}{n}$$




```
ReturnRandomNode(L):

u = head(L)
i = 1      // nodes seen so far
while u ≠ nil:
     r = Random(i)
     if r = 1, x = u
     u = succ(u)
     i = i + 1
return x
```

Events:

$A =$ Algorithm returns v

For $1 \leq i \leq n$:

$A_i = x$ changes during iteration i

$$\Pr(A) = \Pr\left(A_k \wedge \overline{A}_{k+1} \wedge \overline{A}_{k+2} \wedge \cdots \wedge \overline{A}_n\right)$$

$$\Pr(A) = \Pr(A_k) \cdot \Pr\left(\overline{A}_{k+1}\right) \cdot \Pr\left(\overline{A}_{k+2}\right) \cdot \ldots \cdot \Pr\left(\overline{A}_n\right)$$

$$= \frac{1}{k} \cdot \frac{k}{k+1} \cdot \frac{k+1}{k+2} \cdot \ldots \cdot \frac{n-2}{n-1} \cdot \frac{n-1}{n}$$

$$= \frac{1}{\cancel{k}} \cdot \frac{\cancel{k}}{\cancel{k+1}} \cdot \frac{\cancel{k+1}}{\cancel{k+2}} \cdot \ldots \cdot \frac{\cancel{n-2}}{\cancel{n-1}} \cdot \frac{\cancel{n-1}}{n}$$

$$= \frac{1}{n}$$

Long Runs in Random Bitstrings

Want to write a *random bitstring* of length n.

Flip a fair coin n times, where all flips are mutually independent.

If H write 0

If T write 1

A *run* is a sequence of bits that have the same value.

10**11111**01001010

5 1's

6 0's

100101**000000**1

Should we expect to find long runs in random bitstrings?

If we define *long* as $\log n$ bits then yes.

$$\Pr(\text{run of } \log n \text{ bits}) \approx 1 - \frac{1}{n^2}$$

As $n \to \infty$, $\Pr(\text{run of } \log n \text{ bits}) \to 1.$

We should be surprised if there is *not* a run of $\log n$ bits in a long bitstring.

Long Runs in Random Bitstrings

Let R be a random bitstring of length n.

Let $k \leq n$ be an integer.

Event: $A = R$ contains a run of length k

We will find a lower bound on $\Pr(A)$.

Show if k is slightly less than $\log n$, then $\Pr(A) \geq 1 - \frac{1}{n^2}$.

For each $i, 1 \leq i \leq n - k + 1$:

$A_i =$ there is a run of length k starting at i

Example: If $k = 5$, then A_3 occurs in the bitstring below.

1	2	3	4	5	6	7	8	9	10
1	0	**1**	**1**	**1**	**1**	**1**	0	1	0

If $k = 4$, then A_3 and A_4 both occur.

Long Runs in Random Bitstrings

Let R be a random bitstring of length n.

Let $k \leq n$ be an integer.

Event: $A = R \text{ contains a run of length } k$

For each $i, 1 \leq i \leq n - k + 1$:

$A_i = \text{there is a run of length } k \text{ starting}$ at i

$A = A_1 \vee A_2 \vee \cdots \vee A_{n-k+1}$

Thus

$\Pr(A) = \Pr(A_1 \vee A_2 \vee \cdots \vee A_{n-k+1})$

These events are not pairwise disjoint. If A_3 happens, then $\Pr(A_4) = \frac{1}{2}$. Thus we would need *inclusion/exclusion*.

We can try the complement rule:

$$\bar{A} = \bar{A}_1 \wedge \bar{A}_2 \wedge \cdots \wedge \bar{A}_{n-k+1}$$

And

$\Pr(\bar{A}) = \Pr(\bar{A}_1 \wedge \bar{A}_2 \wedge \cdots \wedge \bar{A}_{n-k+1})$.

Long Runs in Random Bitstrings

Let R be a random bitstring of length n.

Let $k \leq n$ be an integer.

Event: $A = R$ contains a run of length k

For each $i, 1 \leq i \leq n - k + 1$:

$A_i =$ there is a run of length k starting at i

$A = A_1 \vee A_2 \vee \cdots \vee A_{n-k+1}$

Thus

$\Pr(A) = 1 - \Pr(\bar{A})$

$\Pr(\bar{A}) = \Pr(\bar{A}_1 \wedge \bar{A}_2 \wedge \cdots \wedge \bar{A}_{n-k+1})$.

We know $\Pr(\bar{A}_i) = 1 - \Pr(A_i)$.

Let $r_1, r_2, \ldots . r_n$ be the bits in R.

$\Pr(r_i = 0) = \frac{1}{2}$ and $\Pr(r_i = 1) = \frac{1}{2}$.

$$A_i = (r_i = r_{i+1} = \cdots = r_{i+k-1} = 1) \text{ or } (r_i = r_{i+1} = \cdots = r_{i+k-1} = 0)$$

These are disjoint events, thus

$$\Pr(A_i) = \Pr(r_i = r_{i+1} = \cdots = r_{i+k-1} = 1) + \Pr(r_i = r_{i+1} = \cdots = r_{i+k-1} = 0)$$

Long Runs in Random Bitstrings

Let R be a random bitstring of length n.

Let $k \leq n$ be an integer.

Event: $A = R \text{ contains a run of length } k$

For each $i, 1 \leq i \leq n - k + 1$:

$A_i = \text{there is a run of length } k \text{ starting at } i$

$$\Pr(A_i) = \Pr(r_i = r_{i+1} = \cdots = r_{i+k-1} = 1) + \Pr(r_i = r_{i+1} = \cdots = r_{i+k-1} = 0)$$

$$= \frac{1}{2^k} + \frac{1}{2^k} = 2 \cdot \frac{1}{2^k} = \frac{1}{2^{k-1}}.$$

$$\Pr(\bar{A}_i) = 1 - \Pr(A_i)$$

$$= 1 - \frac{1}{2^{k-1}}.$$

$A = A_1 \vee A_2 \vee \cdots \vee A_{n-k+1}$

Thus

$\Pr(A) = 1 - \Pr(\bar{A})$

$\Pr(\bar{A}) = \Pr(\bar{A}_1 \wedge \bar{A}_2 \wedge \cdots \wedge \bar{A}_{n-k+1})$.

However, $\bar{A}_1, \bar{A}_2, \ldots, \bar{A}_{n-k+1}$ are not mutually independent.

Long Runs in Random Bitstrings

Let R be a random bitstring of length n.

Let $k \leq n$ be an integer.

Event: $A = R$ contains a run of length k

For each $i, 1 \leq i \leq n - k + 1$:

$A_i =$ there is a run of length k starting at i

$A = A_1 \vee A_2 \vee \cdots \vee A_{n-k+1}$

Thus

$\Pr(A) = 1 - \Pr(\bar{A})$

$\Pr(\bar{A}) = \Pr(\bar{A}_1 \wedge \bar{A}_2 \wedge \cdots \wedge \bar{A}_{n-k+1})$.

Divide R into blocks of length k.

Block1	Block2	Block3	Block4	Block5	Block6	Block7
10110	11001	10000	11111	10100	00000	10101

Let $B_i =$ Block i contains a run of length k.

In the above string with $k = 5$, B_4 and B_6 both occur.

Events $\bar{B}_1, \ \bar{B}_2, \dots, \bar{B}_{n/k}$ are mutually independent, since the blocks do not overlap.

If $\bar{A}$ occurs then $\bar{B}_1 \wedge \bar{B}_2 \wedge \cdots \wedge \bar{B}_{n/k}$ occurs (but not necessarily the converse).

Long Runs in Random Bitstrings

Let R be a random bitstring of length n.

Let $k \leq n$ be an integer.

Event: $A = R \text{ contains a run of length } k$

For each $i, 1 \leq i \leq n - k + 1$:

$A_i = \text{there is a run of length } k \text{ starting at } i$

$A = A_1 \vee A_2 \vee \cdots \vee A_{n-k+1}$

Thus

$\Pr(A) = 1 - \Pr(\bar{A})$

$\Pr(\bar{A}) = \Pr(\bar{A}_1 \wedge \bar{A}_2 \wedge \cdots \wedge \bar{A}_{n-k+1})$.

Thus $\Pr(\bar{A}) \leq \Pr(\bar{B}_1 \wedge \bar{B}_2 \wedge \cdots \wedge \bar{B}_{n/k})$, and

$$\Pr(A) \geq 1 - \Pr(\bar{B}_1 \wedge \bar{B}_2 \wedge \cdots \wedge \bar{B}_{n/k})$$

Since $\bar{B}_1, \bar{B}_2, \ldots, \bar{B}_{n/k}$ are mutually independent:

$\Pr(\bar{B}_1 \wedge \bar{B}_2 \wedge \cdots \wedge \bar{B}_{n/k}) = \Pr(\bar{B}_1) \cdot \Pr(\bar{B}_2) \cdot \ldots \cdot \Pr(\bar{B}_{n/k})$.

$$\Pr(\bar{B}_i) = \Pr(\bar{A}_i) = 1 - \frac{1}{2^{k-1}}$$

Long Runs in Random Bitstrings

Let R be a random bitstring of length n.

Let $k \leq n$ be an integer.

Event: $A = R$ contains a run of length k

For each $i, 1 \leq i \leq n - k + 1$:

$A_i =$ there is a run of length k starting at i

$A = A_1 \vee A_2 \vee \cdots \vee A_{n-k+1}$

$\Pr(A) = 1 - \Pr(\bar{A})$

$$\Pr(\bar{B}_i) = \Pr(\bar{A}_i) = 1 - \frac{1}{2^{k-1}}$$

$$\Pr(\bar{A}) \leq \left(1 - \frac{1}{2^{k-1}}\right)^{n/k}$$

$1 - x \leq e^{-x}$, thus

$$1 - \frac{1}{2^{k-1}} \leq e^{-\frac{1}{2^{k-1}}} = e^{-2/2^k}$$

$\Pr(\bar{B}_1 \wedge \bar{B}_2 \wedge \cdots \wedge \bar{B}_{n/k})$

= $\Pr(\bar{B}_1) \cdot \Pr(\bar{B}_2) \cdot \ldots \cdot \Pr(\bar{B}_{n/k})$.

$$\Pr(\bar{A}) \leq \left(e^{-\frac{2}{2^k}}\right)^{\frac{\mathrm{n}}{\mathrm{k}}} \leq e^{-2n/k2^k}$$

Long Runs in Random Bitstrings

Let R be a random bitstring of length n.

Let $k \le n$ be an integer.

Event: $A = R$ contains a run of length k

For each $i, 1 \le i \le n - k + 1$:

$A_i =$ there is a run of length k starting at i

$A = A_1 \vee A_2 \vee \cdots \vee A_{n-k+1}$

$\Pr(A) = 1 - \Pr(\bar{A})$

$$\Pr(\bar{A}) \le \left(e^{-\frac{2}{2^k}}\right)^{\frac{\mathrm{n}}{\mathrm{k}}}$$

$$\le e^{-2n/k2^k}$$

If we choose $k = \log n - 2 \log \log n$:

$$2^k = 2^{\log n - 2 \log \log n}$$

$$2^{\mathrm{k}} = \frac{2^{\log n}}{2^{2 \log \log n}} = \frac{n}{\log^2 n}$$

Long Runs in Random Bitstrings

Let R be a random bitstring of length n.

Let $k \leq n$ be an integer.

Event: $A = R$ contains a run of length k

For each $i, 1 \leq i \leq n - k + 1$:

$A_i =$ there is a run of length k starting at i

$A = A_1 \vee A_2 \vee \cdots \vee A_{n-k+1}$

$\Pr(A) = 1 - \Pr(\bar{A})$

$$\frac{2n}{k2^k} = \frac{2\log^2 n}{k}$$

$$= \frac{2\log^2 n}{\log n \ - 2\log\log n}$$

$$\geq \frac{2\log^2 n}{\log n} = 2\log n$$

$$= \frac{2\ln n}{\ln 2} \geq 2\ln n$$

Long Runs in Random Bitstrings

Let R be a random bitstring of length n.

Let $k \leq n$ be an integer.

Event: $A = R$ contains a run of length k

For each $i, 1 \leq i \leq n - k + 1$:

$A_i =$ there is a run of length k starting at i

$A = A_1 \vee A_2 \vee \cdots \vee A_{n-k+1}$

$\Pr(A) = 1 - \Pr(\bar{A})$

$$\Pr(\bar{A}) \leq e^{-\frac{2n}{k2^k}}$$

$$\leq e^{-2 \ln n}$$

$$\leq \frac{1}{n^2}.$$

Thus

$$\Pr(A) \geq 1 - \frac{1}{n^2}.$$