Chapter 9

Graphs

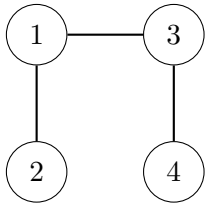
9.1 Graphs

This is an important section, as you'll see graphs A LOT in this course and in the courses to follow.

9.1.1 Definitions, Examples, and Basics

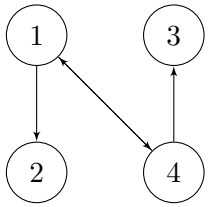
Definition 9.1.1. *A **non-directed graph** G is a pair of two sets - $G = (V, E)$ - V being a set of vertices and E being a set of couples of vertices, which represent edges ("links") between vertices.*

Example: $G = (\{1,2,3,4\}, \{\{1,2\},\{1,3\},\{3,4\}\})$ is the following graph:



Definition 9.1.2. *A **directed graph** G is a pair of two sets - $G = (V, E)$ - V being a set of vertices and $E \subseteq V \times V$ being a set of directed edges ("arrows") between vertices.*

Example: $G = (\{1,2,3,4\}, \{(1,2),(1,4),(4,1),(4,3)\})$ is the following graph (note that it has arrows):



Definition 9.1.3. *A **weighted graph** composed by a graph $G = (V, E)$ (either non-directed or directed) and a weight function $w : E \to \mathbb{R}$. Usually (but not necessary), we will think about the quantity $w(e)$, where $e \in E$, as the length of the edge.*

Now that we see graphs with our eyes, we can imagine all sorts of uses for them... For example, they can represent the structure of the connections between friends on Facebook, or they can even represent which rooms in your house have doors between them.

Remark 9.1.1. *Note that directed graphs are a **generalization** of non-directed graphs, in the sense that every non-directed graph can be represented as a directed graph. Simply take every non-directed edge $\{v, u\}$ and turn it into two directed edges $(v, u), (u, v)$.*

Remark 9.1.2. *Note that most of the data structures we discussed so far - Stack, Queue, Heap, BST - can all be implemented using graphs.*

Now let's define some things in graphs:

Definition 9.1.4. *(Path, circle, degree)*

1. *A **simple path** in the graph G is a series of unique vertices (that is, no vertex appears twice in the series) $v_1, v_2, ..., v_n$ that are connected with edges in that order.*
2. *A **simple circle** in the graph G is a simple path such that $v_1 = v_n$.*
3. *The **distance** between two vertices $v, u \in V$ is the length of the shortest path between them (∞ if there is no such path).*

Remark 9.1.3. *Note that for all $u, v, w \in V$ the triangle inequality holds regarding path lengths. That is:*

$$dist(u, w) \leq dist(u, v) + dist(v, w)$$

Definition 9.1.5. *(connectivity)*

1. *Let $G = (V, E)$ be a non-directed graph. A **connected component** of G is a subset $U \subseteq V$ of maximal size in which there exists a path between every two vertices.*
2. *A non-directed graph G is said to be a **connected** graph if it only has one connected component.*
3. *Let $G = (V, E)$ be a directed graph. A **strongly connected component** of G is a subset $U \subseteq V$ of maximal size in which for any pair of vertices $u, v \in U$ there exist both directed path from u to v and a directed path form v to u.*

Claim 9.1.1. *Let $G = (V, E)$ be some graph. If G is connected, then $|E| \geq |V| - 1$*

Proof. We will perform the following process: Let $\{e_1, ..., e_m\}$ be an enumeration of E, and let $G_0 = (V, \emptyset)$. We will build the graphs $G_1, G_2, ...G_m = G$ by adding edges one by one. Formally, we define -

$$\forall i \in [m] \;\; G_i = (V, \{e_1, ..., e_i\})$$

G_0 has exactly $|V|$ connected components, as it has no edges at all. Then G_1 has $|V| - 1$. From there on, any edges do one of the following:

1. Keeps the number of connected components the same (the edge closes a cycle) '

2. Lowers the number of connected components by 1 (the edges does not close a cycle)

So in general, the number of connected components of G_i is $\geq |V| - i$. Now, if $G_m = G$ is connected, it has just one connected component! This means:

$$1 \geq |V| - |E| \implies |E| \geq |V| - 1$$

□

9.1.2 Graph Representation

Okay, so now we know what graphs are. But how can we represent them in a computer? There are two main options. The first one is by **array of adjacency lists**. Given some graph G, every slot in the array will contain a linked list. Each linked list will represent a list of some node's neighbors. The second option is to store edges in an **adjacency matrix**, a $|V| \times |V|$ binary matrix in which the v, u-cell equals 1 if there is an edge connecting u to v. That matrix is denoted by A_G in the example below. Note that the running time analysis might depend on the underline representation.

Question. What is the memory cost of each of the representations? Note that while holding an adjacency matrix requires storing $|V|^2$ bits regardless of the size of E, Maintaining the edges by adjacency lists costs linear memory at the number of edges and, therefore, only $\Theta(|V| + |E|)$ bits.

Example. Consider the following directed graph:

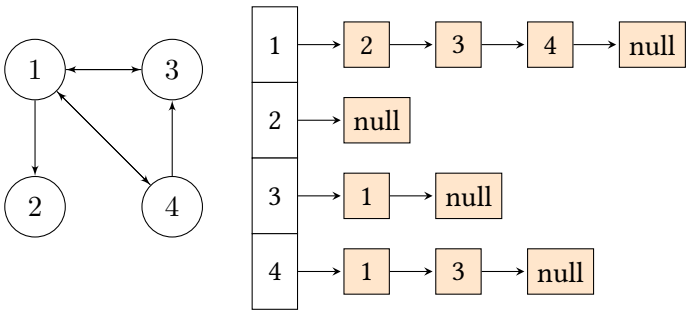


Figure 9.3: Persenting G by array of adjacency lists.

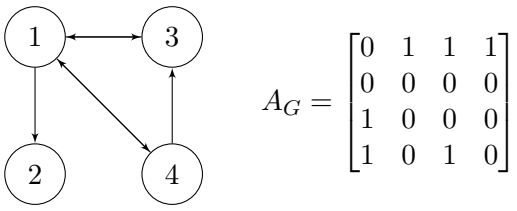


Figure 9.4: Presenting G by adjacency matrix.

9.1.3 Breadth First Search (BFS)

One natural thing we might want to do is to travel around inside a graph. That is, we would like to visit all of the vertices in a graph in some order that relates to the edges. Breadth-first search constructs a breadth-first tree, initially containing only its root, which is the source vertex s. Whenever the search discovers a white vertex v in the course of scanning the adjacency list of a gray vertex u, the vertex v and the edge (u, v) are added to the tree. We say that u is the predecessor or parent of v in the breadth-first tree. Since every vertex reachable from s is discovered at most once, each vertex reachable from s has exactly one parent. (There is one exception: because s is the root of the breadth-first tree, it has no parent.) Ancestor and descendant relationships in the breadth-first tree are defined relative to the root s as usual: if u is on the simple path in the tree from the root s to vertex v, then u is an ancestor of v, and v is a descendant of u. The breadth-first-search procedure BFS on the following page assumes that the graph $G = (V, E)$ is represented using adjacency lists. It denotes the queue by Q, and it attaches three additional attributes to each vertex v in the graph:

1. v.visited is a boolean flag which indicate wheter v was allready visited.
2. $\pi(v)$ is v's predecessor in the breadth-first tree. If v has no predecessor because it is the source vertex or is undiscovered, then $\pi(v)$ is None/NULL.

```
1  for v ∈ V do
2  |  v.visited ← False
3  end
4  Q ← new Queue
5  Q.Enqueue(s)
6  s.visited ← True
7  while Q is not empty do
8  |  u ← Q.Dequeue()
9  |  for neighbor w of u do
10 |  |  if w.visited is False then
11 |  |  |  w.visited← True
12 |  |  |  π(w) ← u
13 |  |  |  Q.Enqueue(w)
14 |  |  end
15 |  end
16 end
```

Algorithm 1: BFS(G, s)

Correctness: The example should be enough to explain the correctness. A concrete proof can be found in the book, page 597.

Runtime: We can analyse the runtime line-by-line:

- Lines 1-2: $|V|$ operations, all in $O(1)$ runtime, for a total of $O(|V|)$.
- Lines 3-6: $O(1)$
- Lines 7-8: First we need to understand the number of times the $while$ loop iterates. We can see that every vertex can only enter the queue ONCE (since

it is then tagged as "visited"), and therefore it runs $\leq |V|$ times. All operations are $O(1)$, and we get a total of $O(|V|)$.

- Lines 9-13: Next, we want to understand the number of times this for loop iterates. The for loop starts iterating once per vertex, and then the number of its iterations is the same as the number of neighbors that this vertex has. Thus, it runs $O(|E|)$ times.

So all in all we get a runtime of $O(|V| + |E|)$

9.1.4 Usage of BFS

Now we have a way to travel through a graph using the edges. How else can we use it?

Exercise: Present and analyse an algorithm $CC(G)$ which receives some undirected graph G and outputs the number of connected components in G in $O(|V|+|E|)$.

Solution: Consider the following algorithm:

```
1 count ← 0
2 for v ∈ V do
3 |   if v.visited = False then
4 |   |   count←count+1
5 |   |   BFS(G, v)
6 |   end
7 end
8 return count
```

Algorithm 2: CC(G)

9.1.5 Depth First Search (DFS)

As its name implies, depth-first search searches "deeper" in the graph whenever possible. Depth-first search explores edges out of the most recently discovered vertex v that still has unexplored edges leaving it. Once all of v's edges have been explored, the search "backtracks" to explore edges leaving the vertex from which v was discovered. This process continues until all vertices that are reachable from the original source vertex have been discovered. If any undiscovered vertices remain, then depth-first search selects one of them as a new source, repeating the search from that source. The algorithm repeats this entire process until it has discovered every vertex.

```
1  DFS( G):
2  for v ∈ V do
3  |  vi.visited ← False
4  end
5  time ← 1
6  for v ∈ V do
7  |  if not v.visited then
8  |  |  π (v) ← null
9  |  |  Explore( G, v )
10 |  end
11 end
```

Algorithm 3: DFS(G)

```
1  Explore(G, v):
2  Previsit(v) for (v, u) ∈ E do
3  |  if not u.visited then
4  |  |  π (u) ← v
5  |  |  Explore( G, u )
6  |  end
7  end
8  Postvisit(v)
```

```
1  Previsit(v):
2  pre(v) ← time
3  time ← time +1
```

```
1  Postvisit (v):
2  post(v) ← time
3  time ← time +1
```

Properties of depth-first search. Depth-first search yields valuable information about the structure of a graph. Perhaps the most basic property of depth-first search is that the predecessor subgraph G_π does indeed form a forest of trees since the structure of the depth-first trees exactly mirrors the structure of recursive calls of explore-function. That is, $u = \pi(v)$ if and only if explore(G, v) was called during a search of u's adjacency list. Additionally, vertex v is a descendant of vertex u in the depth-first forest if and only if v is discovered during the time in which u is gray. Another important property of depth-first search is that discovery and finish times have a parenthesis structure. If the explore procedure were to print a left parenthesis "(u" when it discovers vertex u and to print a right parenthesis r"u)" when it finishes u, then the printed expression would be well-formed in the sense that the parentheses are properly nested.

The following theorem provides another way to characterize the parenthesis structure.

Parenthesis theorem In any depth-first search of a (directed or undirected) graph $G = (V, E)$, for any two vertices u and v, exactly one of the following three conditions holds:

1. the intervals [pre(u), post(u)] and [pre(v), post(v)] are entirely disjoint, and neither u nor v is a descendant of the other in the depth-first forest.

2. the interval [pre(u), post(u)] is contained entirely within the interval [pre(v), post(v)], and u is a descendant of v in a depth-first tree, or

3. the interval [pre(v), post(v)] is contained entirely within the interval [pre(u), post(u)], and v is a descendant of u in a depth-first tree.

Proof. We begin with the case in which pre(u) $<$ pre(v). We consider two subcases, according to whether pre(v) $<$ post(u). The first subcase occurs when pre(v) $<$ post(u), so that v was discovered while u was still gray, which implies that v is a descendant of u. Moreover, since v was discovered after u, all of its outgoing edges are explored, and v is finished before the search returns to and finishes u. In this case, therefore, the interval [pre(v), post(v)] is entirely contained within the interval [pre(u), post(u)]. In the other subcase, post(u) $<$ pre(v), and by defintion, pre(u) $<$ post(u) $<$ pre(v) $<$ post(v), and thus the intervals [pre(u), post(u)] and [pre(v), post(v)] are disjoint. Because the intervals are disjoint, neither vertex was discovered while the other was gray, and so neither vertex is a descendant of the other.

Corollary. Nesting of descendants' intervals. Vertex v is a proper descendant of vertex u in the depth-first forest for a (directed or undirected) graph G if and only if pre(u) $<$ pre(v) $<$ post(v) $<$ post(u).

9.2 Probability Spaces.

Definition 9.2.1. *A probability space is defined by a tuple (Ω, P), where:*

1. *Ω is a set, called the sample space. Any element $\omega \in \Omega$ is an atomic event. Conceptually, we think of atomic events as possible outcomes of our experiment. Any subset $A \subset \Omega$ is an event.*

2. *P, called the probability function, is a function that assigns a number in $[0, 1]$ to any event, denoted as $P : 2^{\Omega} \to [0, 1]$, and satisfies:*

   (a) *For any event $A \subset \Omega$, $P(A) = \sum_{\omega \in A} P(\omega)$.*

   (b) *Normalization over the atomic events to 1, which means $\sum_{\omega \in \Omega} P(\omega) = 1$.*

Example 9.2.1. *Consider a dice rolling, where each of the faces is indexed by $1, 2, 3, 4, 5, 6$ and has an equal chance of being rolled. Therefore, our atomic events are associated with the rolling result, and P is defined as $P(\omega) = \frac{1}{6}$ for any such atomic event. An example of an event can be $A =$ "the dice falls on an even number". The probability of this outcome is:*

$$P(A) = \sum_{\omega \in A} P(\omega) = P(\{2\}) + P(\{4\}) + P(\{6\}) = 3 \cdot \frac{1}{6} = \frac{1}{2}$$

Claim 9.2.1. *The probability function satisfies the following properties:*

1. $P(\emptyset) = 0$.
2. *Monotonicity: If* $A \subset B \subset \Omega$, *then* $P(A) \leq P(B)$.
3. *Union Bound:* $P(A \cup B) \leq P(A) + P(B)$.
4. *Additivity for disjoint events: If* $A \cap B = \emptyset$, *then* $P(A \cup B) = P(A) + P(B)$.
5. *Complementarity: Denote by* $\bar{A}$ *the complementary event of* A, *which means* $A \cup \bar{A} = \Omega$. *Then,* $P(\bar{A}) = 1 - P(A)$.

Example 9.2.2. *Let's proof the additivity of disjointness property. Let* A, B *disjointness events, so* $A \cap B = \emptyset$ *then*

$$\begin{aligned} P(A \cup B) &= \sum_{w \in A \cup B} P(w) \\ &= \overbrace{\sum_{w \in A, w \notin B} P(w)}^{P(A)} + \overbrace{\sum_{w \in B, w \notin A} P(w)}^{P(B)} + \overbrace{\sum_{w \in A, w \in B} P(w)}^{0} \\ &= P(A) + P(B) \end{aligned}$$

Definition 9.2.2. *Let* (Ω, P) *be a probability space. A random variable* X *on* (Ω, P) *is a function* $X : \Omega \to \mathbb{R}$. *An indicator, is a random variable defined by an event* $A \subset \Omega$ *as follows*

$$X(\omega) = \begin{cases} 1 & \omega \in A \\ 0 & \omega \notin A \end{cases}$$

Sometimes, we will use the notation $\{X = x\}$ *to denote the event* A *such:*

$$A = \{\omega : X(\omega) = x\} := \{X = x\}$$

Example 9.2.3. *Consider rolling a pair of dice. Denote by* $X : [6] \times [6] \to [6]$ *the random variable that is set to be the result of the first roll. Let* Y *be defined in almost the same way, but setting the result of the second die. Namely, if we denote by* $\{(i, j)\}$ *the atomic event associated with sample* i *on the first die and* j *on the second die, then:*

$$\begin{aligned} X(\{i, j\}) &= i \\ Y(\{i, j\}) &= j \end{aligned}$$

In addition, one can define the random variable z *as the sum,* $Z = X + Y$. *Since the sum is also a function from* Ω *to* $\mathbb{R}$, Z *is also a random variable. An example of an indicator could be* W, *which gets* 1 *if* $Z \in \{2, 7, 8\}$.

Example 9.2.4. *Let* X *be an indicator of event* A. *Then* $1 - X$ *is the indicator of* $\bar{A}$.

$$1 - X(\omega) = \begin{cases} 0 & \omega \in A \Leftrightarrow \omega \notin \bar{A} \\ 1 & \omega \notin A \Leftrightarrow \omega \in \bar{A} \end{cases}$$

Definition 9.2.3. *We will say that two events A, B are independent if:*

$$P(A \cap B) = P(A) \cdot P(B)$$

Similarly we will say that random variables $X, Y : \Omega \to \mathbb{R}$ are independent if for any $x \in$ Im X and $y \in$ Im Y:

$$P(X = x \cap Y = y) = P(X = x) \cdot P(Y = y)$$

Example 9.2.5. *X, Y defined in Example 9.2.3 are independent.*

$$\begin{aligned} P(\{X = i\} \cap \{Y = j\}) &= \sum_{i'=i \text{ and } j'=j} P(\{(i', j')\}) = P(\{(i, j)\}) \\ &= \frac{1}{36} = \frac{1}{6} \cdot \frac{1}{6} = P(X = i)P(Y = j) \end{aligned}$$

Example 9.2.6. *Let A and B be independent events. Then, $\bar{A}$ and B are also independent events, since:*

$$\begin{aligned} P(B) &= P(B \cap \Omega) = P(B \cap (A \cup \bar{A})) = P((B \cap A) \cup (B \cap \bar{A})) \\ &= P(B \cap A) + P(B \cap \bar{A}) = P(B)P(A) + P(B \cap \bar{A}) \\ \Rightarrow P(B \cap \bar{A}) &= P(B)(1 - P(A)) = P(B)P(\bar{A}) \end{aligned}$$

Example 9.2.7. *Let X and Y be indicators of independent events A and B. Then $P(X \cdot Y = 1) = P(X = 1) \cdot P(Y = 1)$. The proof is left as an exercise.*

9.3 Throwing Keys to Cells.

Example 9.3.1. *Imagine that following experiment, we have m cells and n keys (balls, numbers, or your favorite object type). We throw each of the keys independently into the cells. The cells are identical, so the probability of hitting any of them is the same, $1/m$. We would like to analyze how the capacity of the cells is distributed.*

1. *What is the probability that the first and the second keys will be thrown to the first cell? What is the probability that the first and the second keys will be thrown to the same cell?*

2. *What is the probability that in the first cell there is exactly one key?*

Let us define the indicator X_i^j which indicate that the jth key fallen into the ith cell.

1. *So first we been asked whether $X_1^1 \cdot X_1^2 = 1$, Since this happens only if both $X_1^1 = 1, X_1^2 = 1$ then by independently we have that:*

$$\begin{aligned} P(X_1^1 \cdot X_1^2 = 1) &= P(X_1^1 = 1 \cap X_1^2 = 1) \\ &= P(X_1^1 = 1) \cdot P(X_1^2 = 1) = \frac{1}{m^2} \end{aligned}$$

   Now, to answer if the first and second keys fall into the same cell, we need to check if there exists an i such that $X_i^1 \cdot X_i^2 = 1$. Observes that for any different i and i', the X_i^j and $X_{i'}^j$ are indicators of disjoint events. This is

because j cannot be in both the i and i' cells. Therefore, $X_i^1 \cdot X_i^2$ and $X_{i'}^1 \cdot X_{i'}^2$ are also indicators of disjoint events. Thus:

$$P(\exists i : X_i^1 \cdot X_i^2 = 1) = P(\bigcup_i X_i^1 \cdot X_i^2 = 1)$$
$$= \sum_i P(X_i^1 \cdot X_i^2 = 1) = m \cdot \frac{1}{m^2} = \frac{1}{m}$$

We are basically done. However, we want to present the same calculation in a different notation that will be useful for computing expectations later on. Note that the random variable that counts "how many" cells both the first and the second fall into is $\sum_i X_i^1 \cdot X_i^2$. In other words, the sum can be either 0 if the keys fall into different cells, or 1 if they both fall into the same cell.

2. *The event that only the jth key falls into the first cell matches to*

$$\left\{ X_1^j \prod_{j \neq j'} \left(1 - X_1^{j'}\right) = 1 \right\}$$

Therefore, due to the disjointness of $1 - X_1^{j'}$ and $X_1^{j'}$, the indicator for the first cell containing exactly one key is:

$$\left\{ \sum_j X_1^j \prod_{j \neq j'} \left(1 - X_1^{j'}\right) = 1 \right\}$$

Since the terms in the sum are disjoint and the products are products of independent indicators, we have:

$$P\left(\sum_j X_1^j \prod_{j \neq j'} \left(1 - X_1^{j'}\right) = 1 \right) = \sum_j P\left(X_1^j \prod_{j \neq j'} \left(1 - X_1^{j'}\right) = 1 \right)$$
$$= m \cdot \frac{1}{m} \left(1 - \frac{1}{m}\right)^{n-1} = \left(1 - \frac{1}{m}\right)^{n-1}$$

Definition 9.3.1. *Let $X : \Omega \to \mathbb{R}$ be a random variable, the expectation of X is*

$$\mathbf{E}[X] = \sum_{\omega \in \Omega} X(\omega) P(\omega) = \sum_{x \in \mathit{Im}\, X} x P(X = x)$$

Observes that if P is distributed uniformly, then the expectation of X is just the arithmetic mean:

$$\mathbf{E}[X] = \sum_{\omega \in \Omega} X(\omega) P(\omega) = \frac{1}{|\Omega|} \sum_{\omega \in \Omega} X(\omega)$$

Claim 9.3.1. *The expectation satisfies the following properties:*

1. *Monotonic, If $X \leq Y$ (for any $\omega \in \Omega$) then $\mathbf{E}[X] \leq \mathbf{E}[Y]$.*
2. *Linearity, for $a, b \in \mathbb{R}$ it holds that $\mathbf{E}[aX + by] = a\mathbf{E}[X] + b\mathbf{E}[Y]$.*

3. *Independently, if X, Y are independent, then $\mathbf{E}[X \cdot Y] = \mathbf{E}[X] \cdot \mathbf{E}[Y]$.*

4. *For any constant $a \in \mathbb{R}$ we have that $\mathbf{E}[a] = a$.*

Proof. 1. Monotonic, if $X \leq Y$ then :

$$\mathbf{E}[X] = \sum_{\omega \in \Omega} X(\omega)P(\omega) \leq \sum_{\omega \in \Omega} Y(\omega)P(\omega) = \mathbf{E}[Y]$$

2. Linearity,

$$\begin{aligned}\mathbf{E}[aX + bY] &= \sum_{\omega \in \Omega} (aX(\omega) + bY(\omega))\, P(\omega) \\ &= a \sum_{\omega \in \Omega} X(\omega)P(\omega) + b \sum_{\omega \in \Omega} Y(\omega)P(\omega)\end{aligned}$$

3. Independently,

$$\begin{aligned}\mathbf{E}[XY] &= \sum_{x,y \in \operatorname{Im} X \times \operatorname{Im} Y} xyP(X = x \cap Y = y) \\ &= \sum_{x,y \in \operatorname{Im} X \times \operatorname{Im} Y} xyP(X = x)P(Y = y) \\ &= \sum_{x \in \operatorname{Im} X} \sum_{y \in \operatorname{Im} Y} xyP(X = x)P(Y = y) \\ &= \sum_{x \in \operatorname{Im} X} xP(X = x) \sum_{y \in \operatorname{Im} Y} yP(Y = y) \\ &= \sum_{x \in \operatorname{Im} X} xP(X = x)\mathbf{E}[Y] \\ &= \mathbf{E}[X]\,\mathbf{E}[Y]\end{aligned}$$

4. Let X be the random variable which is also the constant function $X(\omega) = a$ for any $\omega \in \Omega$. Then we have that

$$\begin{aligned}\mathbf{E}[X] &= \sum_{\omega \in \Omega} X(\omega)P(\omega) \\ &= \sum_{\omega \in \Omega} aP(\omega) = a \cdot 1 = a\end{aligned}$$

□

Example 9.3.2. *Let X be an indicator of event A, what are $\mathbf{E}[X]$ and $\mathbf{E}\left[X^2\right]$? Recall that $X(\omega) = 1$ only if $\omega \in A$ and 0 otherwise, thus:*

$$X^k(\omega) = \begin{cases} 1^k = 1 & \omega \in A \\ 0^k = 0 & \textit{else} \end{cases} \Rightarrow X^k(\omega) = X(\omega)$$

Therefore,

$$\mathbf{E}\left[X^k\right] = \sum_{\omega \in \Omega} X^k(\omega)P(\omega) = \sum_{\omega \in \Omega} X(\omega)P(\omega) = \mathbf{E}[X]$$

Example 9.3.3. *Consider the experiment of throwing keys into cells again. What is the expected number of keys that fell into the same cell as the first key? The indicator of the event j and 1 falling into the same cell is given by $\sum_i X_i^1 X_i^j$ and it remains to sum over all the j's. So:*

Note 1: Despite the ease of computing the expectation, calculating the exact probability of $\sum_i \sum_j X_i^1 X_i^j = L$ for some arbitrary L is a difficult task.

$$\begin{aligned}
\mathbf{E}\left[\sum_i \sum_j X_i^1 X_i^j\right] &= \sum_i \sum_j \mathbf{E}\left[X_i^1 X_i^j\right] \\
&= \sum_i \sum_{j\neq 1} \mathbf{E}\left[X_i^1\right]\mathbf{E}\left[X_i^j\right] + \overbrace{\sum_i \mathbf{E}\left[X_i^1\right]}^{j=1} \\
&= m\cdot(n-1)\frac{1}{m^2} + m\cdot\frac{1}{m} = \frac{n-1}{m} + 1
\end{aligned}$$

9.4 Running Time as a Random Variable.

Randomness might appears in algoritmic context in two main cases, in the first the algorithm might behave randomly, means that it flips coins and decided what to do conditioning on the outcomes. In the second case, the input might distributed according to probability function. In both cases the result and running time of the algorithm are random variable. And it's interesting to ask what is the expected running time.

Let us introduce an example for the first case. We are given an array $A_1, A_2, ..., A_n$ and are asked to find the k smallest elements in it. Here, we are going to suggest a random algorithm that is expected to return in linear time, even if we do not make any assumptions about the input, particularly how it is distributed.

Result: returns the k smallest element in $A_1 ... A_n \in \mathbb{R}^n$
1 **if** *left* = *right* **then**
2 | **return** A_{left}
3 **end**
4 pivot ← select random pivot in [left, right]
5 pivot ← partition(A, left, right, pivot)
6 **if** k = *pivot* **then**
7 | **return** A_k
8 **end**
9 **else if** $k <$ *pivot* **then**
10 | right ← pivot - 1
11 **end**
12 **else**
13 | left ← pivot + 1
14 | $k \leftarrow k-$ pivot
15 **end**
16 **return** call select(A, left, right, k)

Algorithm 4: select(A, left, right, k)

Consider the first call to 'select' and let X_m be the indicator for selecting the index of the mth smallest number on line (4). Notice that X_m and the running

time of the recursive calls are independent random variables. Additionally, we will assume in induction that $T(n',k) \leq 2cn'$ for any $n' < n$. Therefore, the expected running time is:

$$\begin{aligned}
T(n,k) &= c\cdot n + \sum_{m<k} X_m \cdot T(n-m, k-m) \\
&\quad + X_k \cdot 1 + \sum_{m>k} X_m \cdot T(m-1,k) \\
\Rightarrow \mathbf{E}\left[T(n,k)\right] &\leq cn + 2c + \sum_{m<k} \mathbf{E}\left[X_m \cdot T(n-m,k-m)\right] \\
&\quad + \sum_{m>k} \mathbf{E}\left[X_m \cdot T(m-1,k)\right] \\
&\leq c\cdot n + 2c + 2c\sum_{m<k} \frac{n-m}{n} + 2c\sum_{m>k}\frac{m-1}{n} \\
&\leq c\cdot n + 2c + 2c\sum_{m<k} \frac{n-m}{n} + 2c\sum_{m>k}\frac{m}{n}
\end{aligned}$$

Claim 9.4.1. *The sum above is maximal when $k = \lfloor n/2 \rfloor$.*

Proof. We will prove that for $k = i+1$, the sum is greater than $k = i$ if $i < \lfloor n/2 \rfloor$. Denote $S_i = \sum_{m<i} \frac{n-m}{n} + \sum_{m>i} \frac{m}{n}$. Then, the substitution of $S_{i+1} - S_i$ becomes:

$$S_{i+1} - S_i = \frac{n-i-1}{n} - \frac{i}{n} = \frac{n-2i-1}{n}$$

And that quantity is positive for any $i < \lfloor n/2 \rfloor$. By symmetry, we obtain that for any $i > \lceil n/2 \rceil + 1$, the quantity $S_i - S_{i+1}$ is positive. Hence, $\lfloor n/2 \rfloor$ is a global maximum. □

Therefore, the expectation is bounded by:

$$\begin{aligned}
&\leq c\cdot n + 2c + 2c \sum_{m<\lfloor n/2 \rfloor} \frac{n-m}{n} + 2c \sum_{m>\lfloor n/2 \rfloor} \frac{m}{n} \\
&= c\cdot n + 2c + 2\cdot 2c \sum_{m>\lfloor n/2 \rfloor} \frac{m}{n} \\
&= c\cdot n + 2c + 2\cdot 2c \cdot \frac{(n/2)\cdot(n/2-1)}{2n} \\
&\leq cn + 2c + n/2 \cdot 2c \leq 2c\cdot n
\end{aligned}$$