CS 260
Design and Analysis of Algorithms
6. Randomized Algorithms

Mikhail Moshkov

Computer, Electrical and Mathematical Sciences & Engineering Division
King Abdullah University of Science and Technology

Randomized Algorithms

We will consider examples of randomized algorithms which can make random decisions during their work. Randomized algorithms are often conceptually much simpler than the deterministic ones.

We will consider algorithms that are always correct, and run efficiently in expectation. We also will consider an algorithm which can give us a correct answer with high probability.

This section contains examples of average-case analysis of algorithms.

Probability

Let S be a *sample space*, which is a set whose elements are called *elementary events*.

An *event* is a subset of S, S is called the *certain* event, and $\emptyset$ is called the *null* event.

Two events A and B are *disjoint* (*mutually exclusive*) if $A \cap B = \emptyset$.

Probability

A *probability distribution* $\Pr\{\}$ on a sample space S is a mapping from events to real numbers such that the following probability axioms are satisfied:

1. $\Pr\{A\} \geq 0$ for any event A.
2. $\Pr\{S\} = 1$.
3. $\Pr\{A \cup B\} = \Pr\{A\} + \Pr\{B\}$ for any two disjoint events A and B. More generally, for any (finite or countably infinite) sequence of events $A_1, A_2, \ldots$ that are pairwise disjoint,

$$\Pr\left\{\bigcup_i A_i\right\} = \sum_i \Pr\{A_i\}.$$

Probability

We have $\Pr\{\emptyset\} = 0$, and $\Pr\{\bar{A}\} = 1 - \Pr\{A\}$ where $\bar{A} = S \setminus A$.

If $A \subseteq B$ then $\Pr\{A\} \leq \Pr\{B\}$.

For any two events A and B,

$$\Pr(A \cup B) = \Pr\{A\} + \Pr\{B\} - \Pr\{A \cap B\}.$$

Probability

A probability distribution is *discrete* if S is a finite or countable infinite sample space. In this case for any event A,

$$\Pr\{A\} = \sum_{s \in A} \Pr\{s\}.$$

If S is finite and

$$\Pr\{s\} = \frac{1}{|S|}$$

for any $s \in S$ then we have the *uniform* probability distribution on S.

Probability

The *conditional probability* of an event A given that another event B occurs is defined to be

$$\Pr\{A|B\} = \frac{\Pr\{A \cap B\}}{\Pr\{B\}}$$

whenever $\Pr\{B\} \neq 0$.

We read $\Pr\{A|B\}$ as the *probability of A given B*.

Probability

Two events are *independent* if $\Pr\{A \cap B\} = \Pr\{A\} \Pr\{B\}$.

A collection of events $A_1, \ldots, A_n$ is *independent* if, for every set of indices $I \subseteq \{1, \ldots, n\}$ we have

$$\Pr\left\{\bigcap_{i \in I} A_i\right\} = \prod_{i \in I} \Pr\{A_i\}.$$

In the general case

$$\begin{aligned} & \Pr\{A_1 \cap \ldots \cap A_n\} \\ = \; & \Pr\{A_1\} \cdot \Pr\{A_2 | A_1\} \cdot \ldots \cdot \Pr\{A_{j+1} | A_1 \cap \ldots \cap A_j\} \cdot \ldots \\ \cdot \; & \Pr\{A_n | A_1 \cap \ldots \cap A_{n-1}\}. \end{aligned}$$

Probability

A (discrete) *random variable* X is a function from a finite or countably infinite sample space S to the real numbers.

For a random variable X and a real number x, we define the event $X = x$ to be $\{s \in S : X(s) = x\}$. Thus,

$$\Pr\{X = x\} = \sum_{s \in S, X(s) = x} \Pr\{s\}.$$

Probability

The function $f(x) = \Pr\{X = x\}$ is the *probability density* function of the random variable X. From the probability axioms, $f(x) \geq 0$ and $\sum_x f(x) = 1$.

The *expected value* (*expectation* or *mean*) of a discrete random variable X is

$$E[X] = \sum_x x \Pr\{X = x\}.$$

which is well defined if the sum is finite or converges absolutely.

Probability

For any two random variables X and Y,

$$E[X + Y] = E[X] + E[Y]$$

if $E[X]$ and $E[Y]$ are defined.

This property is called *linearity of expectation*. It also extends to finite and absolutely convergent summations of expectations.

Selection

Let we have a sequence of n numbers $a_1, \ldots, a_n$. For simplicity, we will assume that all the numbers are distinct. It will be convenient for us to consider this sequence as the set $S = \{a_1, \ldots, a_n\}$.

Let $k \in \{1, \ldots, n\}$. It is required to find the kth largest element in S.

This problem is a generalization of the problem of finding the median.

The *median* is equal to the kth largest element where $k = \frac{n+1}{2}$ if n is odd and $k = \frac{n}{2}$ if n is even.

Selection

We will use the following recursive algorithm $Select(S, k)$. We choose randomly an element $a_i \in S$, the *splitter*, and form two sets: $S^- = \{a_j : a_j < a_i\}$ and $S^+ = \{a_j : a_j > a_i\}$.

If $|S^-| = k - 1$ then the kth largest element is a_i.
If $|S^-| > k - 1$ then the kth largest element lies in S^-, and we recursively call $Select(S^-, k)$.
If $|S^-| = l < k - 1$ then the kth largest element lies in S^+, and we recursively call $Select(S^+, k - 1 - l)$.

It is clear that independently of the choice of the splitter, the considered algorithm returns the kth largest element of S.

Selection

Example 6.1. Selection

$n = 7$, $k = 5$

$S = \{4, 8, 3, 9, 15, 11, 2\}$. The fifth largest element is 9

$S^- = \{3, 2\}$ 4 $S^+ = \{8, 9, 15, 11\}$

$|S^-| = 2 = l < 4 = k - 1$ S^+ $k' = k - 1 - l = 5 - 1 - 2 = 2$

$S^- = \{8, 9\}$ 11 $S^+ = \{15\}$

$|S^-| > k' - 1 = 2 - 1$ S^-

$S^- = \{8\}$ 9 $S^+ = \emptyset$

$|S^-| = k' - 1 = 1$ The answer is 9

Selection

During the construction of S^- and S^+, the considered algorithm makes $n-1$ steps – comparisons of elements (numbers) from S with a_i.

In the worst case, the algorithm chooses the maximal number in S as a splitter, and during the next iteration we will work with S^- such that $|S^-| = n-1$. Thus, for $k=1$ in the worst case the number of steps will be equal to

$$(n-1) + (n-2) + \ldots + 1 = \frac{n(n-1)}{2}.$$

So the worst-case complexity of the considered algorithm is $\Omega(n^2)$.

Selection

Let us try to understand the average-case complexity of the algorithm *Select*.

At the beginning we prove the following statement.

Lemma 6.2. *If we repeatedly perform independent trials of an experiment, each of which succeeds with probability* $p > 0$*, then the expected number of trials we need to perform until the first success is* $\frac{1}{p}$.

Selection

Proof. Let X be the random variable equal to the number of trials. For $j > 0$, we have $\Pr\{X = j\} = (1-p)^{j-1}p$. Then

$$\begin{aligned} E[X] &= \sum_{j=1}^{\infty} j \cdot \Pr\{X = j\} = \sum_{j=1}^{\infty} j \cdot (1-p)^{j-1}p \\ &= \frac{p}{1-p}\sum_{j=1}^{\infty} j \cdot (1-p)^{j} = \frac{p}{1-p} \cdot \frac{(1-p)}{p^2} = \frac{1}{p}. \end{aligned}$$

□

Selection

We will say that the considered algorithm *Select* is in phase j when the size of the set under consideration (we denote it m) satisfies the following inequalities:

$$n\left(\frac{3}{4}\right)^{j+1} < m \leq n\left(\frac{3}{4}\right)^{j}.$$

Let us try to bound the expected number of iterations in phase j (iteration is a choice of splitter and $m-1$ comparisons – steps).

In a given iteration we say that an element of the set is central, if there are at least $\left\lfloor \frac{m}{4} \right\rfloor$ elements which are smaller than it and at least $\left\lfloor \frac{m}{4} \right\rfloor$ elements which are larger than it.

Selection

If a central element will be chosen as a splitter, then at the next iteration we will work with at most $m - \left\lfloor \frac{m}{4} \right\rfloor - 1$ elements (last -1 corresponds to the splitter). It is clear that

$$m - \left\lfloor \frac{m}{4} \right\rfloor - 1 \leq m \left(\frac{3}{4} \right) \leq n \left(\frac{3}{4} \right)^{j+1},$$

i.e., after this iteration the algorithm will be in phase $j + 1$.

Selection

It is clear that the number of central elements is at least $m - 2\left\lfloor \frac{n}{4} \right\rfloor \geq \frac{n}{2}$. So the probability that our random choice of splitters produces a central element is at least $\frac{1}{2}$.

By Lemma 6.2, the expected number of iterations before a central element is found is at most 2. Therefore, the expected number of iterations spent in phase j is at most 2 for any j.

Selection

Let X be a random variable equal to the number of steps (comparisons) taken by the algorithm. We will represent it as a sum $X = X_0 + X_1 + X_2, \ldots$, where X_j is the expected number of steps spent by the algorithm in phase j.

When the algorithm is in phase j, the set has at most $n\left(\frac{3}{4}\right)^j$ elements, and the number of steps required for one iteration is at most $n\left(\frac{3}{4}\right)^j$.

Selection

We know that the expected number of iterations spent in phase j is at most 2, and hence we have $E[X_j] \leq 2n\left(\frac{3}{4}\right)^j$.

Using linearity of expectation we obtain

$$E[X] = \sum_j E[X_j] \leq \sum_j 2n\left(\frac{3}{4}\right)^j = 2n\sum_j \left(\frac{3}{4}\right)^j \leq 8n$$

since the sum $\sum_j \left(\frac{3}{4}\right)^j$ is a geometric series that converges absolutely.

Thus the expected number of steps of *Select* for n-element set is $O(n)$.

Quick Sort

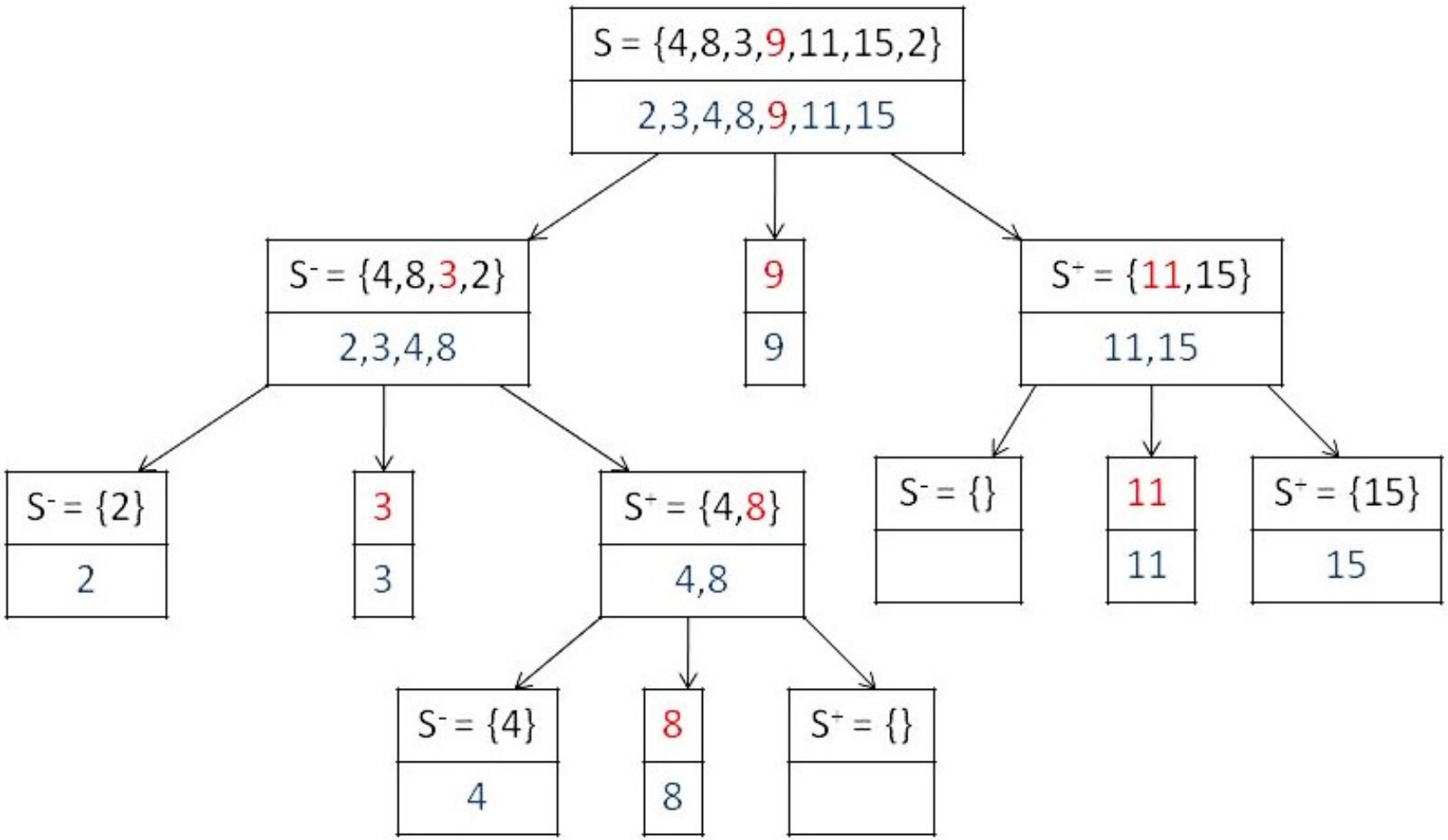
Let we have a sequence of n numbers $a_1, \ldots, a_n$. We should sort this sequence. For simplicity, we will assume that all the numbers are distinct. It will be convenient for us to consider the sequence as the set $S = \{a_1, \ldots, a_n\}$.

We will use the following recursive algorithm $Quicksort(S)$. We choose randomly an element $a_i \in S$, the *splitter*, and form two sets: $S^- = \{a_j : a_j < a_i\}$ and $S^+ = \{a_j : a_j > a_i\}$, and recursively call $Quicksort(S^-)$ and $Quicksort(S^+)$.

We return the sorted set S^-, then a_i, then the sorted set S^+.

Quick Sort

Example 6.3. Quick Sort



Quick Sort

During the iteration (the construction of S^- and S^+) the algorithm makes $n-1$ steps – comparisons of elements from S with the element a_i.

If a_i is the maximal element in S, then $|S^-| = n-1$ and $|S^+| = 0$. If during each iteration the algorithm chooses as the splitter the maximal element, then the algorithm makes

$$(n-1) + (n-2) + \ldots + 1 = \frac{n(n-1)}{2}$$

steps. Thus, the worst-case complexity is $\Omega(n^2)$.

Quick Sort

Let us try to evaluate the average-case complexity (number of steps) for the considered algorithm.

For $t = 1, \ldots, n$, we denote by z_t the tth smallest element in S. We denote by X the random variable which is equal to the number of comparisons which *Quicksort* makes.

Let $i, j \in \{1, \ldots, n\}$ and $i < j$. We denote by X_{ij} the random variable with values from $\{0, 1\}$ which is equal to 1 if and only if *Quicksort* compares z_i and z_j.

Quick Sort

Simple analysis shows that for each i and j *Quicksort* compares z_i and z_j at most one time. Therefore

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}.$$

It is clear that
$E[X_{ij}] = 0 \cdot \Pr\{X_{ij} = 0\} + 1 \cdot \Pr\{X_{ij} = 1\} = \Pr\{X_{ij} = 1\}.$

Using linearity of expectation we have

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{X_{ij} = 1\}.$$

Quick Sort

Now we analyze the event $\{X_{ij} = 1\}$, i.e., the event that z_i and z_j are compared during *Quicksort* run.

Let $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$. *Quicksort* should choose at least one element from Z_{ij} as a splitter. Otherwise, we will not be able to obtain the information that $z_i < z_j$.

Quicksort chooses an arbitrary element from Z_{ij} with the same probability $\frac{1}{j-i+1}$.

Quick Sort

If *Quicksort* chooses z_i or z_j then z_i and z_j will be compared. Otherwise, z_i and z_j will not be compared.

Therefore $\Pr\{X_{ij} = 1\} = \frac{2}{j-i+1}$ and

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = 2 \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{j-i+1}.$$

Quick Sort

It is clear that for $i = 1, \ldots, n$ we have

$$\sum_{j=i+1}^{n} \frac{1}{j-i+1} = \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n-i+1} \leq \sum_{k=2}^{n} \frac{1}{k} \leq \ln n.$$

(It is well known that $\ln k \leq H(k) \leq \ln k + 1$, where $H(k) = 1 + \frac{1}{2} + \ldots + \frac{1}{k}$).

Therefore $E[X] \leq 2n \ln n$ and the expected number of steps of *Quicksort* is $O(n \log n)$.

Global Minimum Cut

Let us consider an undirected graph $G = (V, E)$. A *cut* of G is a partition of V into two nonempty sets A and B.

The *size* of a cut (A, B) is the number of edges with one end in A and the other end in B. A *global minimum cut* (*global min-cut*) is a cut of minimum size.

There exists a polynomial-time algorithm for finding a global min-cut in undirected graphs. We now consider a randomized algorithm with simple structure for min-cut finding. This is Contraction Algorithm which was discovered by David Karger in 1992.

Global Minimum Cut

We will assume that G is a connected undirected *multigraph*, i.e., G can have multiple parallel edges between the same pair of nodes.

For each node u of G, we set $S(u) = \{u\}$. The algorithm chooses an edge $e = (u, v)$ of G uniformly at random and contracts it.

It means that we produce a new graph G' in which u and v are identified into a new node w, $S(w) = S(u) \cup S(v)$.

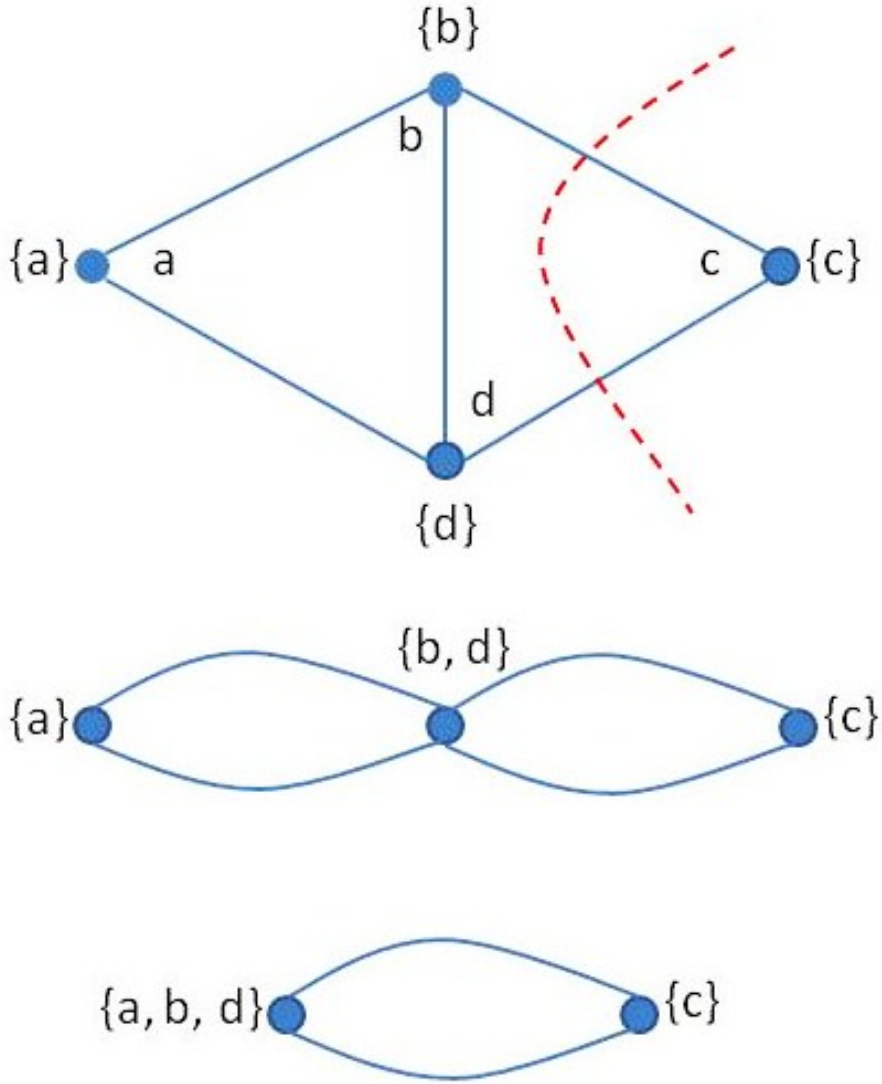
Global Minimum Cut

We remove all edges that connect u and v. Each edge for which exactly one end belongs to $\{u, v\}$ is changed such that this end is equal to w now.

Then the algorithm continues recursively on G'. The algorithm will finish the work when the obtained graph will have exactly two nodes v_1 and v_2. The output of the algorithm is the cut $(S(v_1), S(v_2))$ of G.

Global Minimum Cut

Example 6.4. Contraction algorithm



Global Minimum Cut

Let (A, B) be an arbitrary global min-cut of G.

Theorem 6.5. *The Contraction Algorithm returns (A, B) with probability at least $\frac{2}{n(n-1)}$ where n is the number of nodes in G.*

Global Minimum Cut

Proof. Let the size of (A, B) be equal to k. Since G is connected, $k > 0$. It means that the set F of edges with one end in A and the other end in B contains exactly k edges.

It is clear that the degree of each node in G is at least k (otherwise, the size of global min-cut is less than k). Therefore $|E| \geq \frac{1}{2}kn$. Hence, the probability that an edge in F is contracted during the first iteration is at most

$$\frac{k}{\frac{1}{2}kn} = \frac{2}{n}.$$

Global Minimum Cut

Let us consider now the situation after j iterations when the current graph G' has $n-j$ nodes. Let us assume that no edge in F was contracted during the first j iterations.

One can show that every cut of G' is a cut of G (we should consider each node u in G' as a set $S(u)$ of nodes in G). Therefore the degree of each node in G' is at least k, and the number of edges in G' is at least $\frac{1}{2}k(n-j)$.

So the probability that an edge from F is contracted in the iteration $j+1$ is at most

$$\frac{k}{\frac{1}{2}k(n-j)} = \frac{2}{n-j}.$$

Global Minimum Cut

The cut (A, B) will be returned by the algorithm if no edge of F is contracted in any of iterations $1, 2, \ldots, n-2$.

We denote by I_j the event that an edge of F is not contracted in the iteration j. We have shown that $\Pr\{I_1\} \geq 1 - \frac{2}{n}$ and $\Pr\{I_{j+1} | I_1 \cap I_2 \cap \ldots \cap I_j\} \geq 1 - \frac{2}{n-j}$. We have

$$\begin{aligned}
& \Pr\{I_1 \cap I_2 \cap \ldots \cap I_{n-2}\} = \Pr\{I_1\} \cdot \Pr\{I_2 | I_1\} \cdot \ldots \\
\cdot \quad & \Pr\{I_{j+1} | I_1 \cap I_2 \cap \ldots \cap I_j\} \cdot \ldots \cdot \Pr\{I_{n-2} | I_1 \cap \ldots \cap I_{n-3}\} \\
\geq \quad & \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right) \ldots \left(1 - \frac{2}{n-j}\right) \ldots \left(1 - \frac{2}{3}\right) \\
& = \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right)\left(\frac{n-5}{n-3}\right) \cdots \left(\frac{2}{4}\right)\left(\frac{1}{3}\right) \\
& = \frac{2}{n(n-1)}.
\end{aligned}$$

Global Minimum Cut

Let us try to repeat the Contraction Algorithm $\frac{n(n-1)}{2}$ times. Then the probability that we fail to find a global min-cut in any run is at most

$$\left(1 - \frac{1}{\frac{n(n-1)}{2}}\right)^{\frac{n(n-1)}{2}} \leq \frac{1}{e}.$$

If we run the algorithm $\frac{n(n-1)}{2} \ln n$ times then the probability we fail to find a global min-cut is at most $e^{-\ln n} = \frac{1}{n}$.

Global Minimum Cut

Now we can find the answer for the following question: what is the maximal number of global min-cuts in a connected undirected multigraph $G = (V, E)$ with n nodes.

Let us consider n-node cycle. This graph contains exactly $\frac{n(n-1)}{2}$ global min-cuts obtained by cutting any two edges.

Theorem 6.6. *A connected undirected multigraph $G = (V, E)$ on n nodes has at most $\frac{n(n-1)}{2}$ global min-cuts.*

Global Minimum Cut

Proof. Let $C_1, \ldots, C_r$ denote all global min-cuts of G. Let K_i denote the event that C_i is returned by the Contraction Algorithm, and let $K = \bigcup_{i=1}^{r} K_i$ denote the event that the algorithm returns any global min-cut.

From Theorem 6.5 it follows that $\Pr\{K_i\} \geq \frac{2}{n(n-1)}$. Any two events K_i and K_j are disjoint since only one cut is returned by any given run of the algorithm. Therefore

$$\Pr\{K\} = \Pr\left\{\bigcup_{i=1}^{r} K_i\right\} = \sum_{i=1}^{r} \Pr\{K_i\} \geq \frac{2r}{n(n-1)}.$$

It is clear that $\Pr\{K\} \leq 1$. Therefore $r \leq \frac{n(n-1)}{2}$. □

Global Minimum Cut

It was an example of the situation when randomized construction is used to obtain a result which has nothing to do with randomization.

Such an approach is widely used in combinatorics: one can show the existence of some structure by showing that a random construction produces it with positive probability.

Hashing

Hashing is a technique that allows us to create a dynamically changing set of elements.

Let we have a universe U of possible elements which is extremely large. It can be the set of all possible people, the set of all possible strings in the alphabet $\{0, 1\}$ which length is at most 10000, etc.

Really we will work with a set $S \subseteq U$ which is very small part of U, but we do not know S, moreover S is a dynamically changing set. The goal is to be able to insert and delete elements from S and quickly determine whether a given element belongs to S.

Hashing

It is clear that we cannot create an array with $|U|$ positions – one for each possible element from U since U is too large. We will use another approach.

Let us assume that we will work with sets S such that $|S| \leq n$. We create an array K of size n and use a function $h : U \to \{0, 1, \ldots, n-1\}$ that maps elements of U to array positions.

We call such a function h a *hash function*, and the array K a *hash table*.

Hashing

Of course, it is possible that $h(u) = h(v)$ for some distinct elements $u, v \in S$. We will say that these two elements *collide*.

We will assume that each position $K[i]$ of the hash table contains a linked list of all elements $u \in S$ such that $h(u) = i$.

It is clear that this construction allows us to insert and delete elements from S and determine whether a given element belongs to S.

Hashing

However, we have a serious problem connected with the situation when there exist many elements from S with the same value of the hash function h.

Let $|U| \geq n^2$. Then for each h there exists a set $S \subseteq U$ and a number $i \in \{0, \ldots, n-1\}$ such that $|S| = n$ and for each $u \in S$ we have $h(u) = i$. Our goal now is to show that randomization can help for this.

Hashing

The key idea is to choose a hash function in random from a set H of hash functions which has two properties:

1. For any pair of distinct elements $u, v \in U$ the probability that a randomly chosen $h \in H$ satisfies $h(u) = h(v)$ is at most $\frac{1}{n}$. We say that a class H is *universal* if it satisfies this first property. To use efficiently a universal class we also need H to satisfy a second property.
2. Each $h \in H$ can be compactly represented and, for a given $h \in H$ and $u \in U$, we can compute the value $h(u)$ efficiently.

Hashing

We now explain the notion of universal class of hash functions.

Theorem 6.7. *Let H be a universal class of hash functions, S be an arbitrary subset of U of size at most n, and let u be any element in U. We define X to be a random variable equal to the number of elements $s \in S$ for which $h(s) = h(u)$ for a random choice of hash function $h \in H$. (Here S and u are fixed, and the randomness is in the choice of $h \in H$). Then $E[X] \leq 1$.*

Proof. For an element $s \in S$, we define a random variable X_s which is equal to 1 if $h(s) = h(u)$, and equal to 0 otherwise.

We have $E[X_s] = \Pr\{X_s = 1\} \leq \frac{1}{n}$, since the class H is universal. Now $X = \sum_{s \in S} X_s$. By linearity of expectation, we have

$$E[X] = \sum_{s \in S} E[X_s] \leq |S| \cdot \frac{1}{n} \leq 1.$$

□

Hashing

We now design an universal class of hash functions. We use a prime number $p \geq n$. We assume that U is the set of all vectors $x = (x_1, x_2, \ldots, x_r)$ for some natural r, where x_i is integer and $0 \leq x_i < p$ for each i.

For each $a = (a_1, \ldots, a_r) \in U$, we define the linear function

$$h_a(x) = \left(\sum_{i=1}^{r} a_i x_i \right) \mod p.$$

Hashing

Let $H = \{h_a : a \in U\}$. It is clear that H has the second property. We prove that H is a universal class. We begin with an auxiliary statement.

Lemma 6.8. *For any prime p, any integer z, $z \neq 0 \mod p$, and any two integers α, β, if $\alpha z = \beta z \mod p$, then $\alpha = \beta \mod p$.*

Proof. Suppose $\alpha z = \beta z \mod p$. Then $z(\alpha - \beta) = 0 \mod p$, and $z(\alpha - \beta)$ is divisible by p. But $z \neq 0 \mod p$. So z is not divisible by p.

Since p is prime, $\alpha - \beta$ must be divisible by p. Therefore, $\alpha = \beta \mod p$. □

Theorem 6.9. *The class of linear functions H is universal.*

Proof. Let $x = (x_1, \ldots, x_r)$ and $y = (y_1, \ldots, y_r)$ be two distinct elements of U. We need to show that the probability of $h_a(x) = h_a(y)$, for a randomly chosen $a \in U$, is at most $\frac{1}{p}$.

Hashing

Since $x \neq y$, there exists $j \in \{1, \ldots, r\}$ such that $x_j \neq y_j$. Without loss of generality we assume that $j = 1$. Then $a_1(x_1 - y_1) = \sum_{i=2}^{r} a_i(y_i - x_i) \mod p$.

We denote $z = x_1 - y_1$ and $m = \sum_{i=2}^{r} a_i(y_i - x_i) \mod p$. Therefore $a_1 z = m \mod p$.

Hashing

Let us show that there exists exactly one value a_1 such that $0 \leq a_1 < p$ and $a_1 z = m \mod p$. Let us assume the contrary: there exist two such values a_1 and a'_1.

Then $a_1 z = a'_1 z \mod p$ and, by Lemma 6.8, we have $a_1 = a'_1 \mod p$ and $a_1 = a'_1$ since $0 \leq a_1 < p$ and $0 \leq a'_1 < p$.

Hashing

Therefore, for each $(a_2, \ldots, a_r) \in \{0, 1, \ldots, p-1\}^{r-1}$ there exists at most one $a_1 \in \{0, 1, \ldots, p-1\}$ such that, for $a = (a_1, \ldots, a_r)$, the equality $h_a(x) = h_a(y)$ holds.

So the number of functions h from H for which $h(x) = h(y)$ is at most p^{r-1}. It is clear that $|H| = p^r$.

Hashing

For any pair of distinct elements $x, y \in U$, the probability that a randomly chosen $h \in H$ satisfies $h(x) = h(y)$ is at most

$$\frac{p^{r-1}}{p^r} = \frac{1}{p}.$$

Thus, H has the first property, and H is a universal class. $\square$