KREISE

EULERIAN-TOUR:

- Closed walk that visit every edge once
- $v_{start} = v_{end}$

Theorem: A connected graph $G = (V, E)$ is Eulerian if and only if all vertices in G have even degree.

- Directed graph → has an eulerian-tour if $deg_{in} = deg_{out}$

Hierholzer Algorithm → find eulerian tour by constructing and merging cycles

1) Start from any vertex and build an initial cycle by traversing edges without repetition
   - each visited edge is removed from the graph
   - if all edges are used, the current cycle is the Eulerian tour
2) If unvisited edge remain, find a new cycle
   - locate a vertex in the current cycle that still has unused edges
   - start a new cycle from this vertex
   - Merge this new cycle into the existing one
3) Repeat until all edges are included in a single cycle.

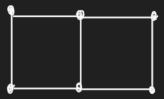
$O(E)$

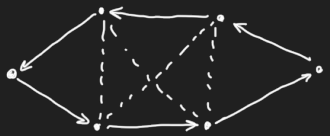
```
EULERTOUR(G, v_start)
// Schneller Läufer
W ← RANDOMTOUR_G(v_start)
// Langsamer Läufer
v^langsam ← Startknoten von W.
while v^langsam ist nicht letzter Knoten in W do
    v ← Nachfolger von v^langsam in W
    if N_G(v) ≠ ∅ then
        W' ← RANDOMTOUR_G(v)
        // Ergänze W = W_1 + ⟨v⟩ + W_2 um die
        // Tour W' = ⟨v'_1 = v, v'_2, ..., v'_{ℓ-1}, v'_ℓ = v⟩
        W ← W_1 + W' + W_2
    v^langsam ← Nachfolger von v^langsam in W
return W
```

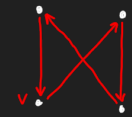
Tips:
- use an adjacency list to efficiently find avaible edges.
- use a stack to traverse edges iterativly and backtrack when necessary

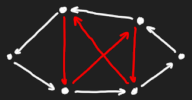
HAMILTONIAN CYCLES

- Cycle in a graph that uses each vertex exactly once
- NP - problem
- A bipartite graph with partitions A and B can only contain a Hamiltonian cycle if $|A| = |B|$
- Dirac's theorem: every graph $G = (V, E)$ with $|V| \geq 3$ and minimal degree $\geq |V|/2$ contains a Hamiltonian cycle
- A grid graph with m rows and n columns contains a Hamiltonian cycle if and only if $m \cdot n$ is even









K-CONNECTIVITY

Def: a graph is k-connected if $|V| \geq k+1$ and $\forall X \subseteq V,\ |X| < k$

Def: a graph is k-edge-connected if $|E| \geq k+1$ and $\forall X \subseteq V, |X| < k$

The graph has at least $k+1$ vertices/edges, and we need to remove at least k vertices/edges to make the graph disconnected

TRAVELLING SALESMAN PROBLEM

! Finding the shortest hamiltonian cycle

- NP-Problem
- There is a 2-approximation algorithm with runtime $O(n^2)$ → Metric TSP

MATCHINGS

Def: Given an undirect graph $G=(V,E)$ a Matching is a set of edges $M \subseteq E$ in which no vertex is incident to more than one edge in M. Formally:

$$e \cap f = \emptyset, \ \forall e,f \in E, \ e \neq f$$

A vertex $v \in V$ is "covered" by M iff there exists an edge $e \in M$ such that $v \in e$

Maximal matching: $M \cup \{e\}$ is not a matching anymore for all edges $e \in E \setminus M$

Maximum matching: $|M| \geq |M'| \ \forall$ matchings M'

} Maximum matching is always a maximal matching

Perfect matching: all vertices are covered in M ($\rightarrow |M| = |V|/2$)

Theorem: Let $G=(A \uplus B, E)$ be a k-regular bipartite graph (↑ every vertex has the same degree). Then there exist matchings $M_1, \dots, M_k$ such that they are pairwise disjoint and construct the entire set of edges together $E = M_1 \uplus \dots \uplus M_k$ and all M_i are perfect matchings.

X: set of vertices of G

$N(X)$: neighbors of X

New graph: $X \uplus N(X)$, the number of edges running between this two sides is some costant number m.

↳ $\sum_{x \in X} \deg(x) = m$ and $\sum_{y \in N(X)} \deg(y) = m \rightarrow \sum_{x \in X} \deg(x) = \sum_{y \in N(X)} \deg(y)$

↳ The graph is k-regular bipartite: $k \cdot |X| = \sum_{x \in X} k = \sum_{x \in X} \deg(x) = \sum_{y \in N(X)} \deg(y) \leq \sum_{y \in N(X)} k = k \cdot |N(X)|$

and thus we have $\boxed{|X| \leq |N(X)|}$

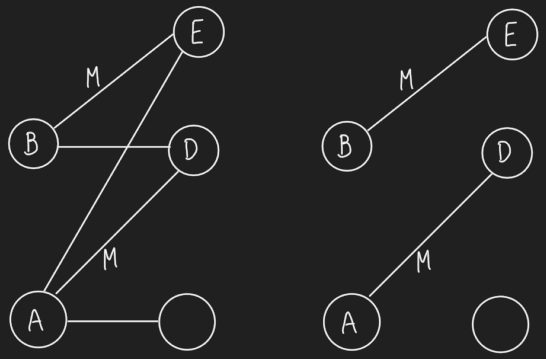
Hall's theorem

Theorem: Let $G = (A \uplus B, E)$ be a 2^k-regular graph, then you can find a perfect matching in $O(|E|)$ time.

HALL'S THEOREM

Theorem: In a bipartite graph $G = (A \uplus B, E)$, there exist a matching of cardinality $|M| = |A|$ if and only if

$$|N(x)| \geq |X| \quad \forall X \subseteq A$$



(i) There exist a matching M, s.t. $|M| = |A| \Rightarrow \forall X \subseteq A$ s.t. $|N(x)| \geq |X|$

(ii) There exist a matching M, s.t. $|M| = |A| \Leftarrow \forall X \subseteq A$ s.t. $|N(x)| \geq |X|$

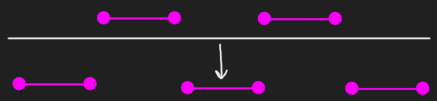
Greedy matching algorithm:

Algorithm 1 GREEDY-MATCHING (G)

```
1: M ← ∅
2: while E ≠ ∅ do
3:     Choose any edge e ∈ E
4:     M ← M ∪ {e}
5:     Remove e and all incident edges in G
6: end while
```

→ Maximal matching

Augmenting path for given matching M: exactly every second edge in the path is in M, starting and ending with vertices not covered by M. This means that an augmenting path has odd length.



We can use an augmenting path to increase the size of our matching by 1 if we remove all the edges along the path from M and add the ones previously not contained (swapping).

Hopcroft-Karp Algorithm

- Work on bipartite graph $G = (A \uplus B, E)$

1) Finding all the augmenting path with minimal length
2) Select an inclusion-maximal set of vertex-disjoint paths and augmenting along all of those at once
3) Repeating until there are no more augmenting paths → Matching is maximum

Runtime: $O(\sqrt{V}\,(|E|+|V|))$

$(|E|+|V|)$: BFS to find augmenting paths

Thearem: Let M be a matching in the graph $G = (V, E)$ M is a maximum matching in G if and only if there exist no M-augmenting path in G.

Blossom's Algorithm

- Works for any graph
- It repeatedly finds augmentings paths and use thos to increase the size of the matching until no more augmenting paths exist → maximum matching
- $O(|E| \cdot |V|^2)$

Hall's Marige Theorem

Theorem (informally): every subset W has enough adjacent vertices in B. Otherwise, we could not pair up the vertices in W with different vertices in B.

↳ Necessary and sufficient condition to have a matching that covers at least one side of the bipartite graph.

ARTICULATION POINTS AND BRIDGES

Articulation points and bridges are vertices and edges that after removing them, the connected components of the graph increases.

Articulation Points and Bridges

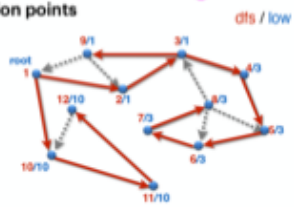
Finding articulation points

Calculate low[v]

low[v] := the smallest dfs-number that one can reach from v with a directed path consisting of (any number of) tree edges and maximum one remaining edge.

Articulation Points and Bridges

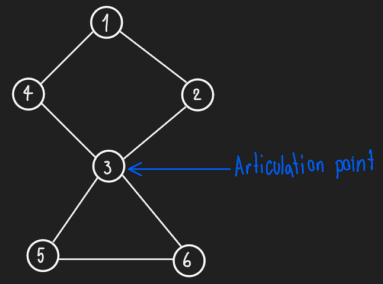
Finding articulation points



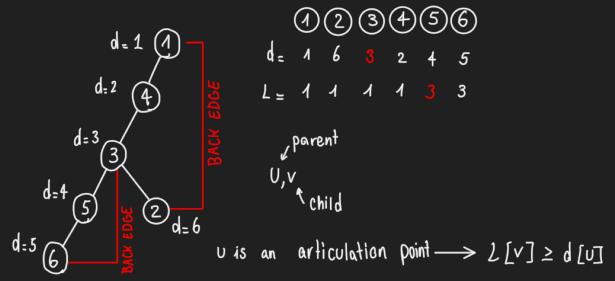
3) Calculate low[v]

$$\text{low}[v] = \min\left(\text{dfs}[v],\ \min_{(v,w)\in E}\begin{cases}\text{dfs}[w], & \text{if (v,w) is a remaining edge}\\ \text{low}[w], & \text{if (v,w) is a tree edge}\end{cases}\right)$$

Example:



DFS - tree:



u is an articulation point $\longrightarrow L[v] \geq d[u]$

A vertex v is an articulation point iff:

1) $v \neq$ root and $L[u] \geq d[v]$
2) v = root and has at least 2 children in DFS-tree

A tree edge e = (v,w) is a bridge iff:

1) $L[u] > d[v]$

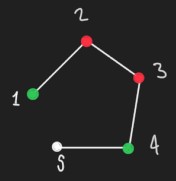
COLORING

Theorem: A graph $G = (V, E)$ is k-partite $\longleftrightarrow$ it can be colored with no more than k colors (i.e $\chi(G) \leq k$).

Greedy algorithm:

- Connected graph → max. degree of a node in G
- It use max $\Delta(G)+1$ colors → complete graph + cycle of odd length → all other graphs $\chi(G) \leq \Delta(G)$ (Brook's theorem)

1) boolean array of length $\deg(v)+1$ ⟶ we track wich colors we had already used for the neighbours of v.
2) we check every adjacent vertices of v ⟶ we set true the vertices that have already a color.
3) we check in the boolean array the first vertex that is false ⟶ since there are max deg(v) neighbours, one of the colour from 0 to deg(v) is avaible
4) the first free color is assigned to v
5) we do the same for all vertices



→ we don't use at most ΔG colors

- $O\left(\sum_v \deg(v)\right) = O(|E|)$

- The algorithm may use more colors than the needed ! ⟶ if the vertices are in the rightorder NO.

- we can check if G is 3-colorable in $O(n^2 \cdot 3^n)$
- Chromatic number = minimum number to color the graph → number of vertices = number of colors
- cycle of odd length are not corable
- every graph on n-nodes is n-colorable
- every tree is bipartite and every k-partite graph is k-colorable

Faster algorithm

- max $\kappa+1$ colors

1) we search the vertex with highest grade and we name it v (if there are more we chose randomly).
2) we delet v and we update the grade of the other vertices $\longrightarrow$ make until the graph is empty.
3) we reset the graph starting from the last deleted vertex, and we assign each node the first avaible color that is not used from the neighbours $\longrightarrow$ since the max degree of every subset is κ, the vertex is going to have max κ-neighbour colored, therefore we need max $\kappa+1$ colors

↓

every time we remove a vertex a new subset is created (the graph without the deleted vertex).

- $O(|V|)$

Theorem: A graph $G=(V,E)$ with $\chi(G)\leq 3$ can be colored in $O(|E|)$ with $O(\sqrt{|V|})$ colors.

PROBABILITY

Discrete probability space: $\Omega = \{w_1, w_2, \ldots\}$

↓

events: each event has a probability $Pr[w_i]$ $(0 \leq Pr[w_i] \leq 1)$ and

$$\sum_{w \in \Omega} Pr[w] = 1$$

Lemma: given two events, A, B, the following holds:

1. $Pr[\emptyset] = 0$, $Pr[\Omega] = 1$
2. $0 \leq Pr[A] \leq 1$
3. $Pr[\bar{A}] = 1 - Pr[A]$
4. If $A \subseteq B$ holds, then $Pr[A] \leq Pr[B]$
5. $Pr[A \cup B] = Pr[A] + Pr[B] - Pr[A \cap B]$

Principle of Inclusion/Exclusion

For events $A_1, A_2, \dots A_n$ $(n \geq 2)$: $$\Pr\left[\bigcup_{i=1}^{n} A_i\right] = \sum_{\ell=1}^{n} (-1)^{\ell+1} \cdot \sum_{1 \leq i_1 < \dots < i_\ell \leq n} \Pr[A_{i_1} \cap \dots \cap A_{i_\ell}]$$

$$= \sum_{i=1}^{n} \Pr[A_i] - \sum_{1 \leq i_1 \leq i_2 \leq n} \Pr[A_{i_1} \cap A_{i_2}] - \dots + (-1)^{n+1} \cdot \Pr[A_1 \cap \dots \cap A_n]$$

If the events are disjoint ($i \neq j$, $A_i \cap A_j = \emptyset$) then: $$\Pr\left(\bigcup_{i=1}^{n} A_i\right) = \sum_{i=1}^{n} \Pr[A_i]$$

$$\Pr\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} \Pr[A_i]$$

Union bound: for events $A_1, \dots, A_n$, we have: $\Pr\left(\bigcup_{i=1}^{n} A_i\right) \leq \sum_{i=1}^{n} \Pr[A_i]$

$$\Pr\left(\bigcup_{i=1}^{\infty} A_i\right) \leq \sum_{i=1}^{\infty} \Pr[A_i]$$

CONDITIONAL PROBABILITY

Definition: Let A and B, be events with $\Pr[B] > 0$. The conditional probability $\Pr[A|B]$ of A given B is:

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

Theorem: Let the events $A_1, \dots, A_n$ be given. If $\Pr[A_1 \cap \dots \cap A_n] > 0$, then the following holds:

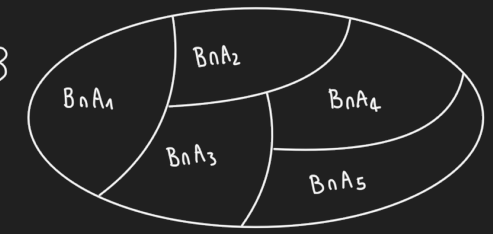
$$\Pr[A_1 \cap \dots \cap A_n] = \Pr[A_1] \cdot \Pr[A_2|A_1] \cdot \Pr[A_3|A_1 \cap A_2] \dots \cdot \Pr[A_n|A_1 \cap \dots \cap A_{n-1}]$$

Law of total probability

Let the events $A_1, \dots, A_n$ be pairwise disjoint and suppose $B \subseteq A_1 \cup \dots \cup A_n$. Then it follows that:

$$\Pr[B] = \sum_{i=1}^{n} \Pr[B \mid A_i] \cdot \Pr[A_i]$$

$$\Pr[B] = \sum_{i=1}^{\infty} \Pr[B \mid A_i] \cdot \Pr[A_i]$$



→ This law let us prove **Bayes' Theorem** (very important!):

Let the events $A_1, \dots, A_n$ be pairwise disjoint. Furthermore, let $B \subseteq A_1 \cup \dots \cup A_n$ be an event with $\Pr[B] > 0$. Then for any $i = 1, \dots, n$ we have

$$\Pr[A_i \mid B] = \frac{\Pr[A_i \cap B]}{\Pr[B]} = \frac{\Pr[B \mid A_i] \cdot \Pr[A_i]}{\sum_{j=1}^{n} \Pr[B \mid A_j] \cdot \Pr[A_j]}$$

INDIPENDENCE

The occurence of one event does not influence the occurence of the other event.

Definition: Two event A and B are indipendent iff:

$$Pr[A \cap B] = Pr[A] \cdot Pr[A]$$

↓

For more than two events we don't need only pairwise indipendence.

Definition: The events $A_1, \dots, A_n$ are indipendent if for all subset $I \subseteq \{1, \dots, n\}$ with $I = \{i_1, \dots, i_k\}$, the following holds: $Pr[A_1 \cap \dots \cap A_{i_k}] = Pr[A_{i_1}] \dots Pr[A_{i_k}]$

Example:

Example 3. *Pairwise Independence ≠ Mutual Independence - Take the same example as before and add another event B*

$A :=$ "First coin is head"

$B :=$ "Second coin is head"

$C :=$ "First and second coin show the same."

As an quick exercise, show that these events are all pairwise independent, but not mutually independent. This is exactly why we need the special definition of mutual independence from above.

Pairwise indipendence:

P(A) = probability of first coin to be head = $\frac{1}{2}$ (head, head), (head, tail)

P(B) = probability of second coin to be head = $\frac{1}{2}$ (head, head), (tail, head)

P(C) = probability of coins to be the same = $\frac{1}{2}$ (head, head), (tail, tail)

$P(A \cap B) = P(\text{head, head}) = \frac{1}{4}$ $P(A) \cdot P(B) = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$ since $P(A \cap B) = P(A) \cdot P(B)$ they're indipendent

Same for the others two

Mutually indipendence:

$P(A \cap B \cap C) = P(\text{head}, \text{head}) = \frac{1}{4}$ but since $P(A) \cdot P(B) \cdot P(C) = \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{8}$ this three events are not mutually indipendent

Lemma: Let A, B and C be indipendent events. Then $A \cap B$ and C, as well as $A \cup B$ and C, are also indipendent.

Proof: The indipendence of $A \cap B$ and C follows from $Pr[(A \cap B) \cap C] = Pr[A] \cdot Pr[B] \cdot Pr[C] = Pr[A \cap B] \cdot Pr[C]$

with the inclusion-exclusion formula,

$$\begin{aligned} Pr[(A \cup B) \cap C] &= Pr[(A \cap C) \cup (B \cap C)] \\ &= Pr[A \cap C] + Pr[B \cap C] - Pr[A \cap B \cap C] \\ &= Pr[C] \cdot (Pr[A] + Pr[B] - Pr[A \cap B]) \\ &= Pr[A \cup B] \cdot Pr[C] \end{aligned}$$

and from that comes the indipendence of $A \cup B$ and C

Lemma: The events $A_1, \dots, A_n$ are indipendent if and only if for all $(s_1, \dots, s_n) \in \{0,1\}^n$, it holds that

$$Pr[A_1^{s_1} \cap \dots \cap A_n^{s_n}] = Pr[A_1^{s_1}] \dots Pr[A_n^{s_n}], \text{ where } A_i^0 = \bar{A}_i \text{ and } A_i^1 = A_i$$

Random Variables

- We want to "simplify" the probabilistic space → **Definition**: A random variable is a function $X: \Omega \to \mathbb{R}$. It's image we will define as $W_X = X(\Omega)$

Example:

Example 5. *Consider $\Omega = [6]^n$ which models the outcomes when we throw a dice n times in a row. Now, let*

$$X_1 : \Omega \to \mathbb{R}$$
$$X_1((a_1, a_2, \ldots, a_n)) = a_1$$

And the set of all events such that the first dice roll is equal to x can now be written as:

$$\{\omega \in \Omega \mid X_1(\omega) = x\}$$

which is still kind of long so we abuse notation and write

$$X_1 = x$$

This notation is indeed a bit sus, but it offers a good middle ground between having to define a billion events of the form A_1, A_2, etc. and having to write out the set of events out in full while offering flexibility. Its power will become apparent when you do more exercises.

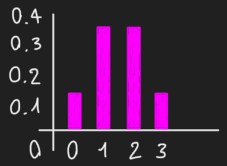
Now that we have introduced random variables, we want to express probabilities in a more orderly way:

Definition: Given a random variable X, there is the distribution (Dichte-Funktion) $f_X(x) = Pr[X = x]$ and then there is the distribution function (Verteilungsfunktion) $F_X(x) = Pr[X \leq x]$

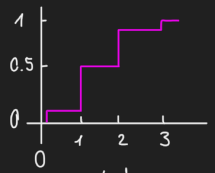
$f_X(x) = Pr[X = x]$: probability that the random variable X assume the value of x.
↳ we use this for specific value

$F_X(x) = Pr[X \leq x]$: probability that the random variable X assume a value equal of less of x.
↳ we use this for intervals

Expectation



distribution



distribution function

The expectation is correlated to the distribution and the distribution function, because is obtained by multiplying each distribution value by the corresponding value and adding the results

Definition: given a random variable X, we define the expectation to be:

$$E[X] := \sum_{x \in W_X} x \cdot Pr[X = x]$$

or equivalently: $E[X] := \sum_{w \in \Omega} X(w) \cdot Pr[w]$ intuitively it is the theoretical average value that we would expect by observing a very large number of realisations on the random variable.

Example:

- Se lanci un dado equo a 6 facce, i possibili risultati sono $\{1, 2, 3, 4, 5, 6\}$ e ciascuno ha probabilità $\frac{1}{6}$.
- L'aspettativa del risultato è:

$$E[X] = 1 \cdot \frac{1}{6} + 2 \cdot \frac{1}{6} + 3 \cdot \frac{1}{6} + 4 \cdot \frac{1}{6} + 5 \cdot \frac{1}{6} + 6 \cdot \frac{1}{6} = 3.5$$

The expected value does not necessarily have to be a value that the random variable can assume

The matematical expectation of a random variable: $E[X] = \sum_{i=1}^{\infty} Pr[X = j]$, if X assumes values in natural numbers ($\mathbb{N}_0$), we can reformulate this sum in an alternative way:

$$E[X] = \sum_{j=0}^{\infty} j \cdot Pr[X = j] = \sum_{j=1}^{\infty} j \cdot Pr[X = j] = \sum_{j=1}^{\infty} \sum_{i=1}^{j} Pr[X = j] \overset{(*)}{=} \sum_{i=1}^{\infty} \sum_{j=i}^{\infty} Pr[X = j] = \sum_{i=1}^{\infty} Pr[X \geq j]$$

This expression is useful because it allows you to calculate the expected value without having to add up directly $j \cdot Pr[X]$. Instead, count "how many times on average" the random variable assumes values greater than or equal to i.

Theorem: Let X be a random variable with $W_X \subseteq \mathbb{N}_0$. Then we have:

$$E[X] = \sum_{i=1}^{\infty} Pr[X \geq i]$$

Theorem: Let $X_1, \dots, X_n$ be arbitrary random variables and $X = a_1 X_1 + \dots + a_n X_n + b$ then we have

$$E[X] = a_1 E[X_1] + \dots + a_n E[X_n] + b$$

↳ If we have more randomised variables $X_1, X_2, \dots, X_n$, and we combine them linearly with coefficients then the expectation of the combination is simply the combination of expectations (b remains unchanged).

Example:

Se lanciamo due dadi X_1 e X_2, il valore atteso della somma è:

$$E[X_1 + X_2] = E[X_1] + E[X_2] = 3.5 + 3.5 = 7$$

Definition: Given an event $A \subseteq \Omega$ we can define the indicator variable for A:

$$\mathbb{I}_A(w) = \begin{cases} 1 \text{ if } w \in A \\ 0 \text{ if } w \notin A \end{cases}$$

→ The variable $\mathbb{I}_A$ indicates it a certain event A it happened or not.

A key observation is that the expectation of $\mathbb{I}_A$ it's just the probability of the event A: $E[\mathbb{I}_A] = Pr[A]$

the variable $\mathbb{I}_A$ assume value 1 with probability $Pr[A]$ and the value 0 with probability $1 - Pr[A]$, so the expected value is: $E[\mathbb{I}_A] = 1 \cdot Pr[A] + 0 \cdot (1 - Pr[A]) = Pr[A]$

Example:

Supponiamo di lanciare un dado e vogliamo sapere se il risultato è un 6.

- Definiamo l'evento $A = \{X = 6\}$.
- La variabile indicatrice vale:
  - $I_A = 1$ se esce 6.
  - $I_A = 0$ altrimenti.
- La probabilità di ottenere un 6 è $\Pr[A] = \frac{1}{6}$.
- Dunque:

$$E[I_A] = \frac{1}{6}$$

Lemma: Given two events $A, B \subseteq \Omega$ we have:

1) $\mathbb{I}_{A \cap B} = \mathbb{I}_A \cdot \mathbb{I}_B$
2) $\mathbb{I}_{\bar{A}} = 1 - \mathbb{I}_A$

CONDITIONAL - / MULTIPLE - / INDIPENDENT - RANDOM VARIABLE:

Definition: Let X be a random variable on Ω, $A \subseteq \Omega$ an event such that $Pr[A] > 0$. $X \mid A$ is what we call a conditional random variable. If $X : \Omega \to \mathbb{R}$, then we have that:

$$X \mid A : A \to \mathbb{R}$$

So the domain of the function X has been reduced to A

Example:

Supponiamo di lanciare un dado a sei facce, quindi:

- $\Omega = \{1, 2, 3, 4, 5, 6\}$
- La variabile aleatoria X è il risultato del lancio: $X(\omega) = \omega$

Ora considera l'evento $A = \{4, 5, 6\}$, cioè "esce un numero maggiore o uguale a 4".

Poiché $\Pr[A] = \frac{3}{6} = 0.5 > 0$, possiamo parlare della variabile aleatoria condizionata $X \mid A$.

Questa nuova variabile:

- Ha dominio ridotto: $A = \{4, 5, 6\}$
- Mantiene gli stessi valori di X, ma **solo su** A

Quindi:

$$X \mid A(4) = 4, \quad X \mid A(5) = 5, \quad X \mid A(6) = 6$$

In questo modo, $X \mid A$ è semplicemente X "visto" solo **quando** sappiamo che è successo l'evento A.

- Conditioned probability distribution: $f_{X|A}(x) = Pr[X = x \mid A]$

  Probability that X assumes the value of x since the event A occured.

- Conditioned distribution function: $F_{X|A}(x) = Pr[X \leq x \mid A]$

  How likely is it that X assumes a value less than or equal to x, since A is occured.

- Conditioned expected value: $E[X|A] = \sum_{x \in W_x} x \cdot Pr[X = x \mid A] = \frac{1}{Pr[A]} \cdot \sum_{w \in A} X(w) \cdot Pr[w]$

  Average of the random variable X, conditioned to the event A.

Example:

Torniamo all'esempio del dado:

- $\Omega = \{1,2,3,4,5,6\}, X(\omega) = \omega$
- Evento $A = \{4,5,6\}$

Distribuzione condizionata:

$$f_{X|A}(x) = \Pr[X = x \mid A] = \begin{cases} \frac{1}{3} & \text{se } x \in \{4,5,6\} \\ 0 & \text{altrimenti} \end{cases}$$

Valore atteso condizionato:

$$\mathrm{E}[X \mid A] = \sum_{x=4}^{6} x \cdot \frac{1}{3} = \frac{4+5+6}{3} = 5$$

Multiple random-variables ⟶ joint probability

- When we have two variable X and Y, and we want to describe the probability that both assume certain values at the same time: $f_{X,Y}(x,y) = \Pr[X=x, Y=y]$ Joint distribution

- Often we are only interested in one variable, indipendently of the other. To do so add up to all the possible values of the other: $f_X(x) = \sum_{y \in W_y} f_{X,Y}(x,y)$ Marginal distribution

Example:

Immagina di lanciare due dadi:

- X: risultato del primo dado
- Y: risultato del secondo dado

Supponendo che siano **dadi perfetti e indipendenti**, la distribuzione congiunta sarà:

$$f_{X,Y}(x,y) = \frac{1}{6} \cdot \frac{1}{6} = \frac{1}{36}$$

La marginale di X sarà:

$$f_X(x) = \sum_{y=1}^{6} f_{X,Y}(x,y) = \sum_{y=1}^{6} \frac{1}{36} = \frac{6}{36} = \frac{1}{6}$$

Idem per Y.

⚠ Two or more variable are indipendent if the joint distribution is factorised as the product of the marginals:

Definition: Let $X_1, \dots X_n$ be random variables. They are called indipendent if and only if:

$$Pr[X_1 = x_1, \dots X_n = x_n] = Pr[X_1 = x_1] \dots Pr[X_n = x_n] \quad \text{for all } x_i \in W_X.$$

↳ When two variables are not indipendent, one can affect the other. In this case, the conditional distribution is used:

$$Pr[X = x \mid Y = y] = \frac{Pr[X = x, Y = y]}{Pr[Y = y]}$$

Indipendent random variable

Two random variable X and Y are indipendent if:

$$Pr[X = x, Y = y] = Pr[X = x] \cdot Pr[Y = y]$$

and far more variable:

$$Pr[X_1 \in S_1, \dots, X_n \in S_n] = \prod_{i=1}^{n} Pr[x_i \in S_i]$$

→

Indicator variable: $\mathbb{I}_A$ and $\mathbb{I}_B$ are indipendent if and only if $f_{X,Y}(1,1) = f_X(1) \cdot f_Y(1)$

we just check only the case where both are 1, because the other values are deterministic.

Note: Let $X_1, \dots, X_n$ be indipendent random variables and $f_1, \dots, f_n : \mathbb{R} \to \mathbb{R}$ real functions, then $f_1(X_1), \dots, f_n(X_n)$ are also indipendent

↳ Example: if X_1 and X_2 are indipendent, then X_1^2 and $\sin(X_2)$ are indipendent.

Theorem: Let X and Y be two indipendent random variable and let $Z := X + Y$, then we have that: $f_Z(z) = \sum_{x \in W_x} f_X(x) \cdot f_Y(z-x)$

↳ Example of sum of variable (sum of two variable with the same distribution → result is the same distribution):

- $\text{Poisson}(\lambda_1) + \text{Poisson}(\lambda_2) = \text{Poisson}(\lambda_1 + \lambda_2)$
- $\text{Bin}(n,p) + \text{Bin}(m,p) = \text{Bin}(n+m, p)$

Theorem (Wald's Equation): Let N and X be two indipendent random variables, where the range of N is the natural numbers and $Z := \sum_{i=1}^{N} X_i$ and $X_1, X_2, \ldots$ are indipent copies of X. Then we have that: $E[Z] = E[N]E[X]$

Example: Hai un numero casuale N di clienti oggi, ognuno spende una quantità aleatoria X_i.
Wald ti dice che **la spesa totale attesa** è:

> numero medio di clienti × spesa media per cliente.

DISTRIBUTIONS:

- A random variable X is said to be a **Bernoulli-Variable** with parameter $0 \leq p \leq 1$ (Bernoulli-verteilt mit Parameter p, $X \sim \text{Bernoulli}(p)$), if and only if we have:

$$f_X(x) = \begin{cases} p \text{ if } x = 1 \\ 1-p \text{ if } x = 0 \\ 0 \text{ else} \end{cases}$$

Example to remember: coin flips!, experiments with binary outcomes in general and all indicator variables.

in which case $E[X] = p$

- A random variable which models multiple indipendent n Bernoulli random variables in a chain (for example multiple coin flips and counting the number of heads). ⟶ A random variable X is said to follow a **binomial distribution** (Binomialverteilt, $X \sim \text{Bin}(n,p)$) if and only if $f_X(x) = \begin{cases} \binom{n}{x} p^x (1-p)^{n-x}, x \in \{0,1,\dots,n\} \\ 0, \text{else} \end{cases}$

and we have $E[X] = p$.

Suppose:
- there are n indipendent trials
- each trials can result in one of two possible outcomes, labelled success and failure
- $P(\text{success}) = p$
- $P(\text{failure}) = 1-p$
- X represents the number of successes in n trials ⟶ then X has a binomial distribution: $P(X=x) = \binom{n}{x} p^x (1-p)^{n-x}$

Why this formula?
$p^x (1-p)^{n-x}$ is the probability of one specific ordering of success and failure and there are $\binom{n}{x}$ orderings.

Example:

According to Statistics Canada life tables, the probability a randomly selected 90 year-old Canadian male survives for at least another year is approximately 0.82.

If twenty 90-year old Canadian males are randomly selected, what is the probability exactly 18 survive for at least another year?

Success = exactly 18 90-years old Canadian male survive for at least one year

Failure = non "

$X \sim \text{Bin}(20, 0,82)$

$P(X=18) = \binom{20}{18} (0,82)^{18} \cdot (1-0,82)^{20-18} = 0,173$ ✓

Example:

A balanced, six-sided die is rolled 3 times.

What is the probability a 5 comes up exactly twice?

Let X represent the number of fives in 3 rolls

X has a binomial distribution with n = 3 and p = 1/6

Success: Rolling a 5

Failure: Rolling anything but a 5

$P(X=2) = \binom{3}{2} \left(\frac{1}{6}\right)^2 \left(1-\frac{1}{6}\right)^{3-2} = 0,0694$ ✓

- Distribution used to model very rare events: poisson-distribution ⟶ A random variable X is said to follow a **poisson distribution** (Poisson-Verteilt, $X \sim Poi(\lambda)$) if and only if:

$$f_X(i) = \begin{cases} \frac{e^{-\lambda} \cdot \lambda^i}{i!} & \text{for } i \in \mathbb{N}_0 \\ 0 & \text{else} \end{cases}$$

- events are occurring independently
- the probability that an event occurs in a given length of time does not change through time

then $E[X] = \lambda$. We have that $\lim_{n \to \infty} Bin(n, \lambda/n) = Poi(\lambda)$

Example:

One nanogram of Plutonium-239 will have an average of 2.3 radioactive decays per second, and the number of decays will follow a Poisson distribution.

What is the probability that in a 2 second period there are exactly 3 radioactive decays?

X = number of decays in a 2 second period

$\lambda = 2{,}3 \cdot 2 = 4{,}6$

$P(X=3) = \frac{e^{-4{,}6} \cdot 4{,}6^3}{3!} = 0{,}163$ ✓

The relationship between the binomial and poisson distribution

The binomial distribution tends toward the poisson distribution as $n \to \infty$, $p \to 0$ and $n \cdot p$ stays constant.

The poisson distribution with $\lambda = n \cdot p$ closely approximates the binomial distribution if n is large and p is small.

⟶ even for a tiny bit of plutoniom, there are very large number of atoms and each one has a tiny probability of experiencing a radioactive decay in a two second period

- **Geometric distribution**: A random variable X is said to be a geometric random variable with parameter p ($X \sim Geo(p)$) if and only if:

$$f_X(i) = \begin{cases} (1-p)^{i-1} \cdot p & \text{for } i \in \mathbb{N} \\ 0 & \text{else} \end{cases}$$

then $E[X] = 1/p$

The cumulative distribution function for the geometric distribution: $F_X(n) = 1 - (1-p)^n$

▶ The geometric distribution is the distribution of the number of trials needed to get the first success in repeated Bernoulli trials.

↳ for the first success to occur on the xth trial:

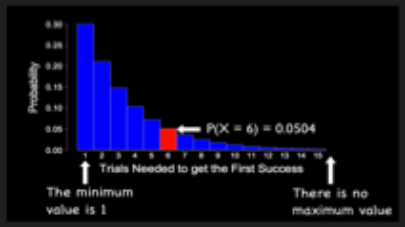
1. The first x-1 trials must be failures $(1-p)^{x-1}$
2. The xth trial must be a success p

Example:

In a large population of adults, 30% have received CPR training. ? RANDOM

If adults from this population are randomly selected, what is the probability that the 6th person sampled is the first that has received CPR training?

$f_X(6) = (1-0,3)^{6-1} \cdot 0,3 = 0,0504$ ✓



The geometric distribution is said to be memoryless:

$\forall s,t \in \mathbb{N}$:

$$Pr[X \geq s+t \mid X > s] = Pr[X > t]$$

$$Pr[X = s+t \mid X = s] = Pr[X = t]$$

↳ intuitive interpretation: if you have already moved on for s failures, the probability that others t failures will be needed are the same as what it takes t fails from zero.

Example:

Supponiamo $p = 0.1$.

Allora:

- $\Pr[X > 2] = (1-0.1)^2 = 0.81$
- $\Pr[X > 5] = (1-0.1)^5 = 0.59049$
- $\Pr[X > 5 \mid X > 2] = \frac{0.59049}{0.81} = 0.729$

Confrontalo con:

- $\Pr[X > 3] = (1-0.1)^3 = 0.729$

✅ Stessa probabilità → proprietà confermata.

- Generalization from one success to multiple successes (waits for the n-th success to stop instaed of the first): **negative binomial distribution**.

A random variable X is said to be a negative binomial random variable with parameter n and p ($X \sim \text{Negative Binomial}(n,p)$) if and only if:

$$f_X(k) = \begin{cases} \binom{k-1}{n-1} \cdot (1-p)^{k-n} \cdot p^n & \text{for } k = 1,2,\dots \\ 0 & \text{else} \end{cases}$$

and then $E[X] = \frac{n}{p}$

- the first $k-1$ trials must result in $n-1$ successes

$$\binom{k-1}{n-1} \cdot p^{n-1} \cdot (1-p)^{(k-1)-(n-1)}$$ ← this is just the binomial formula

- the xth trial must be a success, which has a probability of p

↳ $P(X=x) = p \cdot \binom{k-1}{n-1} \cdot p^{k-1} \cdot (1-p)^{(k-1)-(n-1)} =$

$$= \binom{k-1}{n-1} \cdot (1-p)^{k-n} \cdot p^n$$

Example:

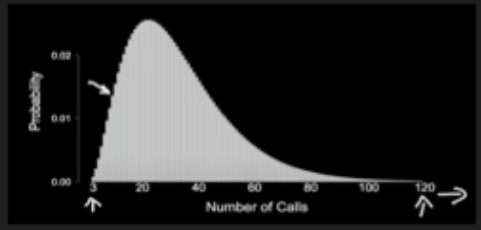
Example: A person conducting telephone surveys must get 3 more completed surveys before their job is finished. → k

On each randomly dialed number, there is a 9% → p chance of reaching an adult who will complete the survey.

What is the probability the 3rd completed survey occurs on the 10th call?

↳ n

$$P(X=10) = \binom{10-1}{3-1} \cdot (1-0{,}09)^{10-3} \cdot (0{,}09)^3 = 0{,}01356 \checkmark$$



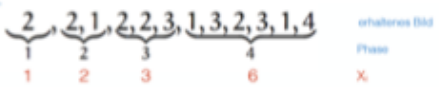
Difference between binomial and negative binomial distribution:

Binomial → number of successes in a fixed number of indipendent Bernoulli trials

Negative binomial → number of trials needed to get a fixed number of successes in repeated indepedent Bernoulli trials

Probability Theory

Coupon Collector



phase i := turns while we have i-1 different coupons

X_i := #turns in phase i

collect all coupons and win

coupon che non ho ancora

$$X_i \sim \text{Geo}\left(\frac{n-(i-1)}{n}\right) \quad E[X_i] = 1/p$$

probabilità di pescarne uno nuovo: $p_i = \frac{n-(i-1)}{n}$

Numero tot. di turni: $X = \sum_{i=1}^{n} X_i$

$$\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}[X_i] = \sum_{i=1}^{n} \frac{n}{n-i+1} = n \cdot \sum_{i=1}^{n} \frac{1}{i} = n \cdot H_n,$$

$H_n = \ln n + \mathcal{O}(1)$

Example to remember :

n different coupons , we're getting one in each turn

X = #turns until we get all n coupons

→ Numero di prove che servono fino al primo successo, dove ogni prova è indipendente e ha una probabilità p di successo

Nel nostro caso:

- Ogni pescata è una prova.
- Il "successo" è trovare un coupon **che non hai ancora**.
- La probabilità di successo cambia **a ogni fase**: se hai $i-1$ coupon, ce ne restano $n-(i-1)$ da trovare, quindi:

$$p_i = \frac{n-(i-1)}{n}$$

Quindi la variabile casuale X_i, che rappresenta **quanti tentativi ti servono per ottenere il prossimo coupon nuovo**, segue:

$$X_i \sim \text{Geo}(p_i)$$

Wichtige Verteilungen

Name	Bezeichnung	Wertebereich	Dichte	Erwartungswert	Varianz
Bernoulli	Bernoulli(p)	$\{0,1\}$	$f_X(i) = \begin{cases} p & \text{für } i = 1, \\ 1-p & \text{für } i = 0. \end{cases}$	p	$p(1-p)$
Binomial	Bin(n,p)	$\{0,1,\dots,n\}$	$f_X(i) = \binom{n}{i} p^i (1-p)^{n-i}$	np	$np(1-p)$
Geometrisch	Geo(p)	$\mathbb{N}$	$f_X(i) = p(1-p)^{i-1}$	$\frac{1}{p}$	$\frac{1-p}{p^2}$
Poisson	Po(λ)	$\mathbb{N}_0$	$f_X(i) = \frac{e^{-\lambda}\lambda^i}{i!}$	λ	λ

VARIANCE

When we study a random variable X, the expected value $E[X]$ tells us the "centre" of its distribution. But with the variance we know how far the values are on average from that centre.

- **Definition**: Let X be a random variable with expectation $\mu = E[X]$. We then define the variance of X to be

  $\mathrm{Var}[X] := E[(X-\mu)^2]$ and we call the square root of this value the standard deviation $\sigma = \sqrt{\mathrm{Var}[X]}$

- With some algebra we can rewrite the variance formula: $\mathrm{Var}[X] = E[X^2] - E[X]^2$

- $\mathrm{Var}[a \cdot X + b] = a^2 \cdot \mathrm{Var}[X]$ the traslation b does not affect the variance because it does not change how much values oscillate, it only moves everything in block.

- Theorem: Let $X_1, \dots, X_n$ be n mutually independent random variables and X equal to the sum of all of those random variables. Then we have that: $\mathrm{Var}[X] = \mathrm{Var}[X_1] + \dots + \mathrm{Var}[X_n]$

Exercise:

Exercise: (Doublesums)

We have blue and red beads in a pot and want to build a necklace with n beads. To do this, we randomly draw a bead from our pot and add it to the necklace. We assume that we have an infinite supply of beads and that each draw results in a red bead with probability 0.25 and a blue bead with probability 0.75. After adding n beads in sequence to our necklace (where the beads are numbered in order), we close it into a loop. We now ask how many color transitions occur in the necklace, meaning how many positions exist where the i-th bead has a different color than the $i+1$-th bead (where the $n+1$-th bead is the same as the 1st bead, since it forms a loop). Let

$$X := \text{Number of color transitions in the necklace}$$

To start, we first define indicator variables for all subproblems (which is almost always a good approach) to simplify the problem. One can verify (or look it up in a reference) that X does not follow a binomial distribution.

$$X_i := \text{Indicatorvariable for the event "There is a color transition at the i-th bead"}$$

$X_j :=$ j-th bead

(a) What is $\mathbb{E}[X]$?

(b) What is $\mathbb{E}[X_i \cdot X_j]$ for arbitrary $1 \le i \le j \le n$? (Hint: Case Distinction)

(c) Using (b), what is $\mathrm{Var}[X]$?

(a)

$$E[X_i] = Pr[\text{transition}] = Pr[\text{red}+\text{blue}] + Pr[\text{blue}+\text{red}] = \frac{1}{4}\cdot\frac{3}{4} + \frac{3}{4}\cdot\frac{1}{4} = \frac{6}{16} = \frac{3}{8}$$

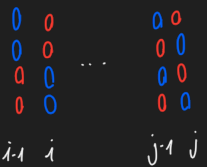
$$E[X] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} \frac{3}{8} = n\cdot\frac{3}{8} \checkmark$$

(b) $i = j \rightarrow E[X_i^2] = E[X_i] = \frac{3}{8}$

$i \neq j$ with i and j adjacent $\rightarrow$ we have that $i \in \{j-1, j+1\}$

$$E[X_i \cdot X_j] = \frac{1}{4}\cdot\frac{3}{4}\cdot\frac{1}{4} + \frac{3}{4}\cdot\frac{1}{4}\cdot\frac{3}{4} = \frac{3}{64} + \frac{9}{64} = \frac{12}{64} \checkmark$$

$i \neq j$ with i and j not adjacent



$$E[X_i \cdot X_j] = E[X_i]\cdot E[X_j] = \frac{3}{8}\cdot\frac{3}{8} = \frac{9}{64}$$

(c) $\mathrm{Var}[X] = E[X^2] - E[X]^2 = E[X_i \cdot X_j] - E[X]^2 =$

$$= E\left[\sum_{i=1}^{n}\sum_{j=1}^{n} X_i \cdot X_j\right] - E[X]^2 = \sum_{i=1}^{n}\sum_{j=1}^{n} E[X_i \cdot X_j] - E[X]^2 =$$

$= ?$

INEQUALITIES

Markov theorem: Let X be a random variable which only attains non-negative values. Then we have that for all $t \in \mathbb{R}^{>0}$: $\Pr[X \geq t] \leq \frac{E[X]}{t}$

Chebyshev theorem: Let X be a random variable and $t > 0$. Then we have: $\Pr[|X - E[X]| \geq t] \leq \frac{\text{Var}[X]}{t^2}$

⚠ If in an exercise we can't get better bounds, we check if our random variable is a binomial random variable, because then we can apply

Chernoff theorem: Let $X_1, \dots, X_n$ be n indipendent Bernoulli random variables such that $\Pr[X_i = 1] = p_i$ and $\Pr[X_i = 0] = 1 - p_i$. Then for the sum of those indipendent random variables $X = X_1 + \dots + X_n$ we have:

- $\Pr[X \geq (1+\delta) \cdot E[X]] \leq e^{-\frac{\delta^2}{3} E[X]}$ for all $0 < \delta < 1$ — X just over the average
- $\Pr[X \geq (1-\delta) \cdot E[X]] \leq e^{-\frac{\delta^2}{2} E[X]}$ for all $0 < \delta < 1$ — X much smaller than the average
- $\Pr[X \geq t] \leq 2^{-t}$ for $t \geq 2e\,E[X]$ — X much bigger

Exercise:

"The probability that something rare happening cannot be greater than this value"

Exercise: (Continuation of Doublesums)

(d) Find a best possible upper bound for $Pr[X \geq t + \mathbb{E}[X]]$ via Markov.

(e) Find a best possible upper bound for $Pr[X \geq t + \mathbb{E}[X]]$ via Chebyshev.

(d)

Let X be a random variable with E[X] = 5 and Var[X] = 0.9. What is the best upper bound for Pr[X ≥ 6] which you can derive?

Chebyshev $\Pr[|X - E[X]| \geq t] \leq \frac{\text{Var}[X]}{t^2}$

$\Pr[X \geq 6] = \Pr[|X - 5| \geq 1] \leq \frac{0,9}{1^2} \approx \frac{9}{10}$

RANDOM ALGORITHMS

LAS-VEGAS ALGORITHM:

Theorem: Let A be an algorithm which never gives a wrong result but sometimes return "???", such that $\Pr[A(I) \text{ correct}] \geq \varepsilon$ (input of the algorithm). Then for all $\delta \geq 0$ let A_δ be the algorithm which repeats A until a non-"???" value is returned or until at most $N = \varepsilon^{-1} \ln \delta^{-1}$ repetitions have occurred (in which case A_δ gives up and returns "???" too). Then, A_δ has the correctness probability: $\Pr[A_\delta(I) \text{ correct}] \geq 1 - \delta$

Proof: the probability that "???" is returned N times (and A_δ thus also returns some nonsense), is

$$\begin{aligned}\Pr[A_\delta(I) \text{ incorrect}] &= (1-\varepsilon)^N \\ &\leq e^{-\varepsilon \varepsilon^{-1} \ln(\delta^{-1})} \\ &= e^{\ln(\delta)} \\ &= \delta\end{aligned}$$

Randomized Algorithms

Reducing the error probability - Las Vegas

Las-Vegas (Runtime is the RV)

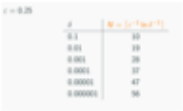
- can output true answer
- cannot output false answer
- can run forever/ can output no answer (???)

To reduce the error probability by a constant factor, only a constant number of additional iterations are necessary

Let A be a randomized algorithm that never returns an incorrect answer but sometimes outputs '???' where $\Pr[A(I) \text{ correct}] \geq \varepsilon$

A_δ: invokes A until either a value different from '???' is returned or '???' is returned $N = \lceil \varepsilon^{-1} \ln \delta^{-1} \rceil$ times

Then it holds that $\Pr[A_\delta(I) \text{ correct}] \geq 1 - \delta$



MONTE CARLO ALGORITHM:

The result is not always correct. However, under certain assumptions, we can improve via the following algorithms

Theorem: Let A be a randomized algorithm which gives a binary output: Either "Yes" or "No", where we have a "one-side" error, i.e.:

- $\Pr[A(I) = \text{"Yes"}] = 1$ if I is a "Yes-Instance" (so the algorithm should return "Yes")
- $\Pr[A(I) = \text{"No"}] = 1$ if I is a "No-Instance" (so the algorithm should return "No")

Then for $\delta > 0$ let A_δ be an algorithm which repeats A until either "No" is returned (in which case A_δ returns "No") or until $N = \varepsilon^{-1} \cdot \ln(\delta^{-1})$ iterations have passed, which all resulted in a "Yes". Then we have for all inputs I: $\Pr[A_\delta(I) \text{ correct}] \geq 1 - \delta$

Theorem: Let $\varepsilon > 0$ and A be a randomized algorithm which either returns "Yes" or "No". Where, indipendent of the instance, we have $\Pr[A(I) \text{ correct}] \geq \frac{1}{2} + \varepsilon$

Then we have for all $\delta > 0$ Let A_δ be the algorithm that repeats $N = 4\varepsilon^{-2} \cdot \ln(\delta^{-1})$ iterations of A and outputs the majority of answers, then we have that $\Pr[A_\delta(I) \text{ correct}] \geq 1 - \delta$

OPTIMIZING A RANDOM ALGORITHM: the returned value should be larger than some baseline $f(I)$. We may now repeat this algorithm until we find a value bigger than $f(I)$:

Theorem: Let $\varepsilon > 0$ and A be a randomized algorithm for an optimization problem, where we have

$$\Pr[A(I) \geq f(I)] \geq \varepsilon$$

then for all $\delta > 0$ let A_δ be the algorithm which repeats A $N = \varepsilon^{-1} \cdot \ln(\delta^{-1})$ times and returns the best result of all iterations. Then we have for A_δ that $\Pr[A_\delta(I) \geq f(I)] \geq 1 - \delta$

Randomized Algorithms

Las-Vegas vs. Monte-Carlo

Las-Vegas

Runtime is the RV

- can output true answer
- cannot output false answer
- can run forever/ can output no answer (???)

Monte-Carlo

Correctness/Quality is the RV

- can output true answer
- can output false answer
- always outputs an answer

Input: An array of $n \geq 2$ elements, in which half are '*a*'s and the other half are '*b*'s.

Output: Find an '*a*' in the array.

```
findingA_LV(array A, n)
begin
    repeat
        Randomly select one element out of n elements.
    until 'a' is found
end
```

```
findingA_MC(array A, n, k)
begin
    i := 0
    repeat
        Randomly select one element out of n elements.
        i := i + 1
    until i = k or 'a' is found
end
```

Target-shooting algorithm:

Target-Shooting

Problem Description

given: finite sets S and U with $S \subseteq U$

U is very large. We cannot afford to iterate through U

to find: $\approx \frac{|S|}{|U|}$ Number of element (|s|) in U

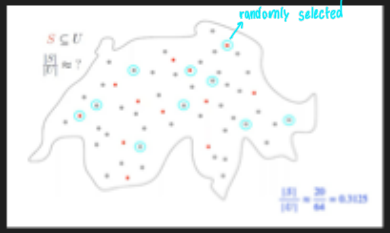
We can generate elements u in U uniformly distributed

$I_S : U \to \{0,1\}$

$I_S(u) = 1 \iff u \in S$

→ Solution: we pick $U_1, \dots, U_N$ from U randomly, uniformly and indipendently and we return: $\frac{1}{N} \cdot \sum_{i=1}^{N} I_S(U_i)$

Example:

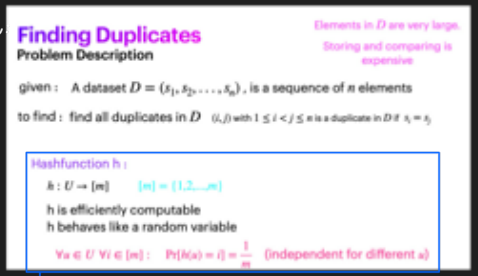


$\frac{1}{10} \cdot \sum_{i=1}^{10} I_S(U_i) = \frac{1}{10} \cdot 3 = \frac{3}{10}$

$\text{Variance} = \frac{1}{N} \cdot \left(\frac{|S|}{|U|} - \left(\frac{|S|}{|U|} \right)^2 \right)$

Hashing: technique used often to speed up datastructure and processes.

Example



↳ $s, t \in U$, then $s = t \Rightarrow h(s) = h(t)$

Hashing solution: we map each element to a number using a hash function but since our m is much smaller than $|U|$ (compression) may happen a collision ($h(s) = h(t)$ but $s \neq t$) and to adjust that, as always we adjust parameters until the probability of collision is very low:

$$\Pr[K_{ij} = 1] = \begin{cases} \frac{1}{m} & \text{if } s_i \neq s_j \\ 0 & \text{else} \end{cases} \Rightarrow E[K_{ij}] \le \frac{1}{m}$$

↳ number of collision: $K = \sum_{1 \le i \le j \le n} K_{ij}$, we apply linearity of expectation and get $E[K] \le \binom{n}{2} \frac{1}{m} \le 1$ for $m = n^2$

Runtime: $O(n \log(n))$ ↓ space of hashing much bigger

Bloom filter solution: this solution has no duplicated but "repeated entry"

Definition: Let $m, k \in \mathbb{N}$ and $h_1, \dots, h_k : U \to [m]$ be k random hash functions and M a boolean array of length m initially set to all 0's.
Then, a bloom filter operates as follows:

1. Iterate through the array of elements D
2. For each element $s \in D$, calculate the hash vector $v = (h_1(s), \dots h_k(s))$
3. For each entry $h_i(s) \in v$ of the hash vector, check if $M[h_i(s)]$ is set to 1. If not, set to 1.
4. If all $h_i(s)$ was set to 1, then add s to the list of repeated entries L

↳ with this method we can't have false negative, only false positive

FINDING PRIME NUMBERS:

Prababilistic solution ⟶ disc. math

Theorem: If n is prime, then $\mathbb{Z}_n$ together with addition and multiplication modulo n is a field. In particular, $x^2 \equiv 1 \pmod{n}$ has only the solution $n-1$ and 1.

Theorem: If n is prime then for all $0 < a < n$, we have $a^{(n-1)} \equiv 1 \pmod{n}$

MILLER-RABIN-PRIMZAHLTEST(n) $O(\ln(n))$

```
1: if n = 2 then
2:     return 'Primzahl'
3: else if n gerade oder n = 1 then
4:     return 'keine Primzahl'
5: Wähle a ∈ {2,3,...,n−1} zufällig und
6: berechne k, d ∈ ℤ mit n − 1 = d2^k und d ungerade.
7: x ← a^d (mod n)
8: if x = 1 or x = n − 1 then
9:     return 'Primzahl'
10: repeat k − 1 mal
11:     x ← x^2 (mod n)
12:     if x = 1 then
13:         return 'keine Primzahl'
14:     if x = n − 1 then
15:         return 'Primzahl'
16: return 'keine Primzahl'
```

we choose this randomly

we will never find proof that x is composite

the first theorem is violated

FINDING ARBITRARILY LONG PATHS:

Given a graph G and a number B we want to find it there exist a path of length at least B in G

Hamiltonian cycle ⟶ NP-complete problem ⟶ Long path too

Costruzione di G':

1. Scegli un nodo arbitrario $v \in G$.
2. Rimuovi v da G.
3. Per ogni **arco incidente in v** (cioè da v a qualche altro nodo), ad esempio $\{v, w_1\}, \{v, w_2\}, \ldots, \{v, w_{deg(v)}\}$, **inserisci nuovi nodi** $\widetilde{w_1}, \widetilde{w_2}, \ldots, \widetilde{w_{deg(v)}}$.
4. Collega ogni nodo nuovo $\widetilde{w_i}$ al rispettivo w_i. Questo sostituisce ogni collegamento da v a un nodo w_i con un nuovo nodo intermedio.

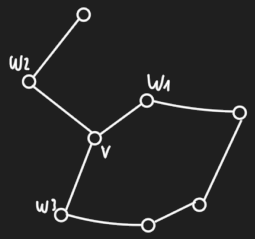
Nota: Abbiamo rimosso un nodo (v) e aggiunto al massimo $deg(v)$ nuovi nodi. Quindi:

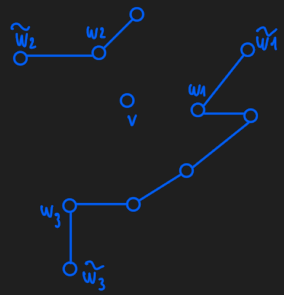
- Numero di nodi totali in G': $(n-1) + deg(v) \leq 2n - 2$

Inoltre:

- Tutti i nuovi nodi $\widetilde{w_i}$ hanno **grado 1** in G'.

$G = (V, E)$ $\qquad$ $G' = (V', E')$





Dimostrazione di equivalenza tra i due problemi: ⟹

(i) Se G ha un ciclo hamiltoniano, allora G' ha un cammino di lunghezza n:

- Supponi che $\langle v_1, v_2, \ldots, v_n, v_1 \rangle$ sia un ciclo hamiltoniano in G, e che $v_1 = v$ (senza perdita di generalità).
- Dopo la trasformazione, i nodi $v_2, v_3, \ldots, v_n$ restano in G', mentre v è stato rimosso.
- Al suo posto, ci saranno due nuovi nodi $\widetilde{v_2}$ e $\widetilde{v_n}$, che collegano v ai suoi vecchi vicini.
- Quindi $\langle \widetilde{v_2}, v_2, v_3, \ldots, v_n, \widetilde{v_n} \rangle$ è un **cammino di lunghezza n** in G'.

(ii) Se G' ha un cammino di lunghezza n, allora G ha un ciclo hamiltoniano: ⟸

- Il cammino in G' è $\langle u_0, u_1, \ldots, u_n \rangle$.
- In G', i soli nodi di grado 1 sono quelli nuovi (i $\widetilde{w_i}$). Quindi u_0 e u_n devono essere due nodi nuovi diversi.
- Il resto del cammino passa per gli $n-1$ nodi originari di G (senza v).
- Quindi $u_1 = \widetilde{w_i}$, $u_{n-1} = \widetilde{w_j}$, e nel mezzo abbiamo tutti gli altri nodi di G.
- Questo corrisponde in G al ciclo $\langle v, u_1, u_2, \ldots, u_{n-1}, v \rangle$, ricostruendo il ciclo hamiltoniano.

SHORT PATH OF LENGTH K:

Does exist a simple path of length at least K ?, where K is relatively small, such as log(n)

↳ COLOR CODING:

Each vertex is assigned a color using a function $v: V \to [K]$, each vertex $v \in V$ is assigned a color $\gamma(v) \in \{1, 2, \dots, K\}$

Colorful path: A path $v_1 \to v_2 \to \cdots \to v_r$ is called colorful path if all its vertices have distinct colors: $\gamma(v_i) \neq \gamma(v_j)$ for all $i \neq j$

We now want to model an alg. to decide if there is any path of length K-1 (i.e K vertices with K diff. colors)

Definition: DP-Definition: Let $P_i(v)$ be our DP entry defined as:

$$P_i(v) := \left\{ S \in \binom{[k]}{i+1} \mid \exists \text{ a colorful path which ends in } v \text{ and traverses exactly the colors in } S \right\}$$

$\exists$ path of length K-1 $\iff \bigcup_{v \in V} P_{K-1}(v) \neq \emptyset$

Definition:

- Base Case 1: $P_0(v) = \{\{\gamma(v)\}\}$, since a path of 0 edges is just the single vertex v and thus only contains the color $\gamma(v)$.
- Recursion: $P_i(v) = \bigcup_{x \in N(v)} \{R \cup \{\gamma(v)\} \mid R \in P_{i-1}(x) \text{ and } \gamma(v) \notin R\}$

  which intuitively means that we range over all possible neighbors of v and append the color of v to the colors of the paths ending in the neighbors of length one less, as long as that path does not contain our current color.

- $\bigcup_{x \in N(v)}$ we look at every neighbour x of v
- we look for a path of length P_{i-1} that ends in x and has colors in R AND that v is not in R

Bunt(G, i):

for all $v \in V$

  $P_i(v) \leftarrow \emptyset$

  for all $x \in N(v)$

    for all $R \in P_{i-1}(x)$ with $\gamma(v) \notin R$

      $P_i(v)$ $P_i(v) \cup \{R \cup \{\gamma(v)\}\}$

$\Rightarrow$ for all i: $O\left(\sum_{i=0}^{K-1} \binom{K}{i} \cdot i \cdot m\right)$

$= O(m \cdot K \cdot 2^K)$

for $K = \log(n) \Rightarrow O(m \cdot \log(n) \cdot n)$

- If we find a colorful path of length $k-1$ → exist a simple path of length $k-1$ in the graph
  - ↳ if we don't: no simple path of length $k-1$ exist or we got unlucky when coloring randomly
- what is the probability of getting lucky (coloring a path of length $k-1$ with k distinct colors)?

Let P (if it exist) be a simple path of length $k-1$ in G

- P can be colored in total in k^k ways (each vertex has k chaices for colors)
- only $k!$ of these are coloring where all nodes have distinct color (i.e colorful)
- therefore the probability that a random coloring makes P colorful is:

$$P[P \text{ is colorful}] = \frac{k!}{k^k} \geq e^{-k}$$

thus, the total probability of success is:

$$P[\text{success}] = P[\text{there exist a colorful path of length } k-1] \geq P[\text{is colorful}] \geq \frac{k!}{k^k} \geq \underbrace{e^{-k}}_{\text{taylor series}}$$

For each iteration of our Monte Carlo algo we have an error probability of at most $1-e^{-\lambda}$ for some λ of our choice. If we want a total error probability of at most e^{-k}, then we apply Theorem 2.74 with $\varepsilon = e^{-k}$ and $\delta = e^{-\lambda}$ and get that we have to repeat this algorithm $N = \lambda \cdot e^k$ times, resulting in a total runtime of:

$$O(N \cdot 2^{\log(n)} \cdot \log(n) \cdot m) = O(\lambda \cdot 2^{\log(n)} \cdot \log(n) \cdot m) = O(\lambda (2e)^{\log(n)} \cdot \log(n) \cdot m)$$

$P[\text{success}] \geq e^{-k} \Rightarrow P[\text{failure}] \leq 1 - e^{-k}$ we repeat that algo $\lambda \cdot e^k$ times $\Rightarrow P[\lambda \cdot e^k \text{ failure}] \leq (1-e^k)^{\lambda e^k}$

Important for the exam:

$$(1-x)^{\frac{1}{x}} < \frac{1}{e} \quad \text{for } x \in (0,1)$$

proof: $\lim_{x \to 0^+} (1-x)^{\frac{1}{x}} = \frac{1}{e}$ (analysis 1)

$f(x) = (1-x)^{\frac{1}{x}}$ is str. $\nearrow$ on $(0,1) \Rightarrow (1-x)^{\frac{1}{x}} < \lim_{x \to 0^+} (1-x)^{\frac{1}{x}} = \frac{1}{e}$ □

for $x = e^{-k} \Rightarrow (1-e^{-k})^{e^k} < \frac{1}{e} \quad | \ (\ )^\lambda$

$\Rightarrow (1-e^{-k})^{\lambda e^k} < \left(\frac{1}{e}\right)^\lambda = e^{-\lambda}$

FLOW

Definition: A Network is a tuple $N(V,A,c,s,t)$ where:

- (V,A) is a direct graph
- $s \in V$ is the source
- $t \in V \setminus \{s\}$ is the target
- $c: A \to \mathbb{R}_0^+$ is the capacity function

Definition: Let $N=(V,A,c,s,t)$ be a network. A flow in N is defined as a function $f: A \to \mathbb{R}$, such that:

- For all $e \in A$, $0 \leq f(e) \leq c(e)$
- For all $v \in V \setminus \{s,t\}$ we have

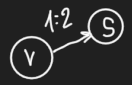
$$\sum_{v \in V \text{ s.t. } (v,v) \in A} f(v,v) = \sum_{v \in V \text{ s.t. } (v,v) \in A} f(v,v)$$

which we call conservation of flow (this does not apply for the source and the target).

Definition: The value of a flow f is defined as:

$$val(f) := netoutflow(s) := \sum_{v \in V \text{ s.t. } (s,v) \in A} f(s,v) - \sum_{u \in V \text{ s.t. } (u,s) \in A} f(u,s)$$

Example:

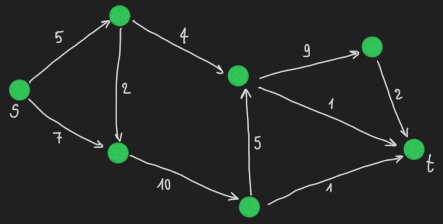


$$val(f(v,s)) = -val(f(s,v)) = -1$$

Definition: f is called an integer flow, iff $f(e) \in \mathbb{Z} \ \forall e \in A$

Lemma:

$$netinflow(t) := \sum_{u \in V \text{ s.t. } (u,t) \in A} f(u,t) - \sum_{v \in V \text{ s.t. } (t,v) \in A} f(t,v) = netoutflow(s)$$



Proof: $0 = netoutflow(s) - netinflow(t)$

$$\left(\sum f(s,u) - \sum f(u,s)\right) - \left(\sum f(u,t) - \sum f(u,s)\right) =$$

$$= \sum_{v \in V} \left(\sum f(v,u) - \sum f(u,v)\right) = \{ \text{Follow from the fact that the net-out and inflow values of all vertices } v \text{ excepts } s \text{ and } t \text{ are } 0.$$

$$= \sum_{(v,u) \in A} f(v,u) - \sum_{(u,v) \in A} f(u,v) = \{ \text{Instead of summing over all ingoing/outgoing edges, we sum over all vertices and count all ingoing/outgoing edges}$$

$$= 0$$

Definition: An **s-t-Cut** for a network (V, A, c, s, t) is a partition (S, T) of V ($S \uplus T = V$) such that $s \in S$ and $t \in T$.

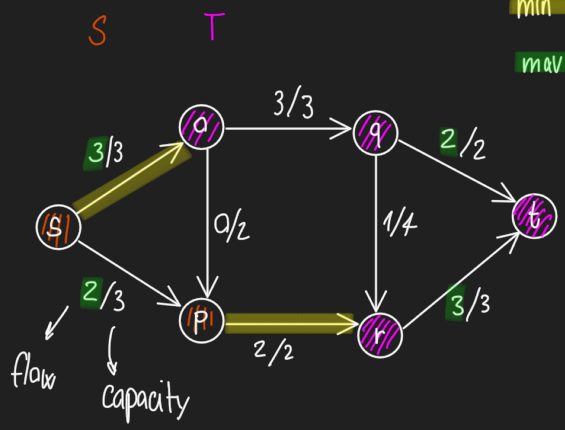
The capacity of this cut is defined as:

$$cap(S,T) := \sum_{(u,w) \in (S \times T) \cap A} c(u,w)$$

Lemma: Let f be a flow and (S,T) an s-t-Cut in a network. Then we have that $val(f) \leq cap(S,T)$

Definition: **Max flow - Min cut - Theorem**: For every network (V, A, c, s, t), we have that:

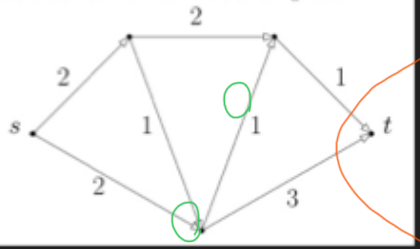
$$\max_{f \text{ a flow in } N} val(f) = \min_{(S,T) \text{ a s-t-Cut in } N} cap(S,T)$$



$\min cap(S,T) = 3+2 = 5$

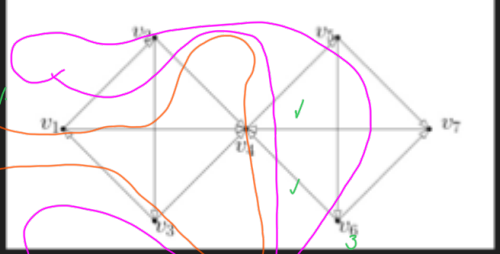
$\max val(f) = 3+2 = 5$

- What is the maximum flow in the following network?



- What is the capacity of the following network with $S = \{v_1, v_2, v_3, v_4\}$, $T = \{v_5, v_6, v_7\}$

- What is the capacity of the following network with $S = \{v_1, v_4, v_5, v_6\}$, $T = \{v_2, v_3, v_7\}$



FLOW ALGORITHM

Given an exisisting flow, we have four ways of changing the flow at a vertex v in hopes of increasing the total flow:

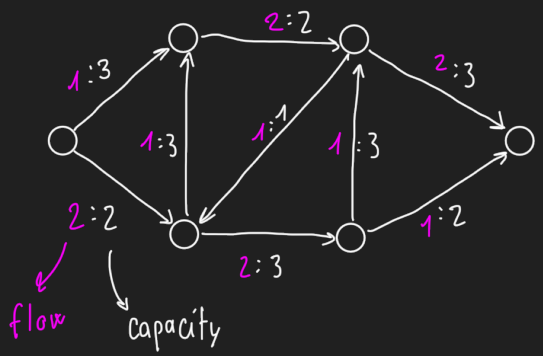
Definition: Let $N = (V, A, c, s, t)$ be a network without edges in opposite directions and f be a flow in N. The residual network N_f is defined as (V, A_f, r_f, s, t), where

- If $e \in A$ with $f(e) < c(e)$ in the original network, then e is also an edge in A_f in the residual network with $r_f(e) := c(e) - f(e)$
- If $e \in A$ with $f(e) > 0$, then there is the edge e^{opp} in A_f with $r_f(e^{opp}) := f(e)$

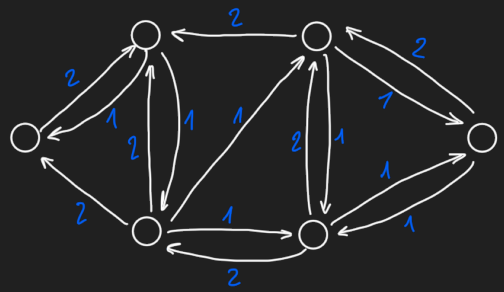
There are no other edges in the network. We call $r_f(e)$ the residual capacity of e.

Example of a residual network:

Original network:



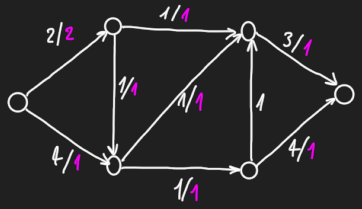
Residual network:



Every s-t path in the residual network corresponds to an augmenting path in the original network. We will augment by a path by finding the smallest residual capacity ε and augmenting the flow and increasing it along that path.

Theorem: A flow f in a network N is a maximum flow if and only if there are no s-t-paths in the residual network N_f anymore

Proof:



$\Leftarrow$: if there is an s-t augmenting path, then we can augment and get bigger flow

$\Rightarrow$: show that there exist a partition (S, T) which has the same capacity as $val(f)$ (thus showing f is maximum), we do this by construction:

- Take S as the set of vertices reachable from s in the residual network. By assumption, $t \notin S$.
- If we take $T = V \setminus S$, then (S, T) is a valid s-t-cut

For this cut, we have that the total flow from S to T is exactly the capacity $cap(S,T)$ and that the total flow from T to S is 0. This gives us (based on the lemma that

$$val(f) = f(S,T) - f(T,S) = cap(S,T) - 0 = cap(S,T)$$

FORD-FULKERSON ALGORITHM

Algorithm 1 Ford-Fulkerson(V, A, c, s, t)

1: $f \leftarrow 0$
2: **while** $\exists$ s-t-path P in (V, A_f) **do**
3: Increase the flow along P (find smallest residual capacity along P, augment)
4: **end while**
5: **return** f

Theorem: Let U be the maximum capacity over all edges in a network N. The runtime of Ford Fulkerson in an integer network is $O(mnU)$ in this network N

Proof: Each iteration of Ford-Fulkerson will take $O(m)$ due to the construction of the residual network. We will have at most $O(nU)$ iterations, as each iteration we increase the flow by at least an integer $\epsilon > 0$ and our flow is upper bounded by $O(nU)$ (the maximum number of outgoing edges s can have is n).

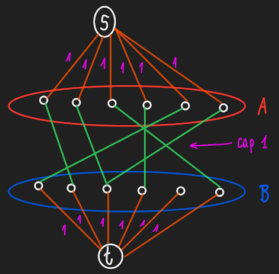
FINDING BIPARTITE MATCHINGS

Given a bipartite graph, we would like to find the maximum matching (Kardinalitätsmaximal).

↓

Given a bipartite graph $G = (U \dot\cup W, E)$, we can construct the network $N = (V, A, c, s, t)$, where

- $V = U \dot\cup W \dot\cup \{s, t\}$
- $A = E \cup (U \times \{s\}) \cup (W \times \{t\})$
- $c(e) = 1 \quad \forall e \in A$



And the maximum flow f_{max} gives us our maximum matching. We can find the edges of that matching, by taking only the edges through which our flow "flows" through. $val(f_{max})$ gives us the size of the maximum matching.

Menger's theorem - Edge Version

Let $G=(V,E)$ be a finite undirected graph and let $s,t \in V$, $s \neq t$.

Then the maximum number of edge disjoint s-t paths is equal to the minimum number of edges whose removal disconnects s from t

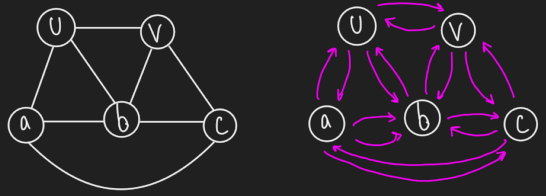
Formally:

$$\max\{\text{edge-disjoint } s\text{-}t \text{ paths}\} = \min\{\text{edges separating } s \text{ and } t\}$$

Furthermore, G is k-edge-connected if and only if for every pair of distinct vertices $s,t \in V$, there exist at least k-edge disjoint s-t paths

$\iff$ removing fewer than k edges never disconnects G.

Proof: Let $G=(V,E)$ be an undirected graph and $N_G=(V,A,c,u,v)$, $A=E \cup E^{opp}$ (G is undirected) and $c(e)=1 \ \forall e \in A$



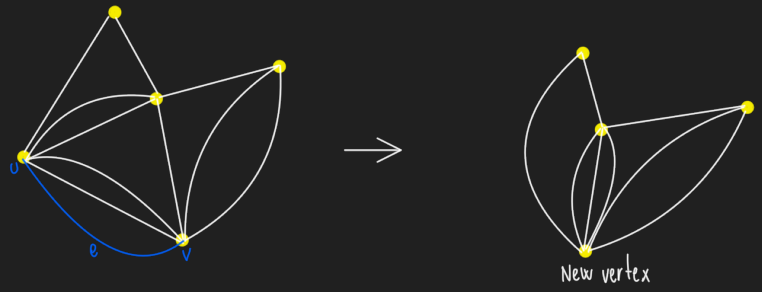
Now, if we compute max-flow in N_G (intuition: it will be the max. no. of edge disjoint paths $u \rightsquigarrow v$).

But we know that max-flow = min-cut (i.e the min no. we have to remove in order to disconnect N_G).

MIN-CUT

We say that a set $C \subseteq E$ is a cut in G if and only if $G' = (V, E \setminus C)$ is disconnected. We are interested in finding a cut of minimum cardinality, a so called min cut. With $\mu(G)$ we denote the size of such a minimum cut in G.

Edge contraction: Let G be a multigraph and let $e = \{u, v\}$ be an edge of G. The contraction of e means that we "glue" u and v together into a single new vertex, removing the edges between u and v, but keeping all other edges incident to u, v, simply redirecting them to the new vertex. The resulting graph is denoted by G/e



LEMMA: Let G be a multigraph and e an edge of G. Then $\mu(G/e) \geq \mu(G)$. Moreover, if there exists a minimum cut C in G such that $e \notin C$, then $\mu(G/e) = \mu(G)$.

↳ contracting an edge can never decrease the size of a minimum cut.

LEMMA: Let $G=(V,E)$ be a multigraph with n vertices. Then the probability that we preserve the size of a min cut $\mu(G) = \mu(G/e)$ for a uniformly randomly chosen edge $e \in E$ is at least: $\Pr[\mu(G) = \mu(G/e)] \geq 1 - \frac{2}{n}$

BASIC VERSION:

- Perform an edge contraction in $O(n)$
- choose an edge uniformly at a random among all edges of the current multigraph in $O(n)$
- find the number of edges connecting two given vertices in $O(1)$

Cut(G):

```
1 while G has more than two vertices do
2     e ← u.a.r. edge from G
3     G ← G/e
4 return number of edges left over
```

The algorithm contract randomly edges until two vertices are left. The edges left that are connecting the last two vertices $= \mu(G)$.

In general the algorithm is correct if and only if:

- $\mu(G) = \mu(G/e)$ for the first contracted edge
- cut succedes for G/e

wich gives us the following recurrence relation for the success probability $p(n)$ (defined as the worst-case success probability for graphs with n vertices):

$$p(n) \geq \frac{2}{n \cdot (n-1)}$$

Increasing success with Monte Carlo algorithm

The probability that all executions will fail is at most:

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

BOOTSTRAPPING

Better algorithm $\longrightarrow$ first iteration has $\frac{2}{n}$ probability to fail, last one has $\frac{2}{3}$ $\longrightarrow$ we do less contraction (for example until the graph has t nodes, where t has to be choosen carefully), then we call again this algorithm. $\longrightarrow$ Runtime: $O(n^2 \cdot \log^k(n)) \approx O(n^2)$

SMALLEST ENCLOSING CIRCLE

Definition: Given a set of points P in the plane with $|P| = n$ (i.e. $P = \{p_1, \dots, p_n\}$, $p_i = (x_i, y_i) \in \mathbb{R}^2$) the smallest enclosing circle problem asks to find a circle $C(P)$ with minimum possible radius r such that all points are contained in this circle

Naive algorithm:

CompleteEnumeration:

```
1 for all Q ∈ (P 3) do
2     compute C(Q)
3     if P ⊆ C(Q)
4         return C(Q)
```

We go over all possible sets Q, compute the enclosing circle in costant time and check if contains all points, if so return it.

Runtime: $O(n^4)$

Randomized algorithm:

```
1 while true
2     pick Q ∈ (P 11) uniformly at random
3     compute C(Q)
4     if P ⊆ C(Q)
5         return C(Q)
6     double all points outside C(Q)
```

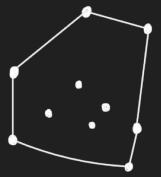
We pick points randomly, every time we find a point that is outside of the circle (we know with higher probability that it will be on the border) we duplicate it, increasing the probability of picking it in the future

Runtime: $O(n \cdot \log(n))$

CONVEX HULL

Definition: A set S is called convex if and only if, for every two points p_1, p_2 contained in the set S, all the points on the straight line connecting the two points are also included in the set S. Formally:

$$S \text{ convex} \longrightarrow \forall\, p_1, p_2 \in S, t \in [0,1] : p_1 + t \cdot (p_2 - p_1) \in S$$



JARVIS WRAP ALGORITHM

1. Select the point p_0 that lies on convex hull (e.g. lowest x-coordinate)
2. Find next point p_{i+1}, such that all other points lie left of the line going through p_i, p_{i+1}
3. Repeat until you reach p_0 again

JarvisWrap(P)

```
h ← 0
p_now ← Punkt in P mit kleinster x-Koordinate
repeat
    q_h ← p_now
    p_now ← FindNext(q_h)
    h ← h + 1
until p_now = q_0
return (q_0, q_1, ..., q_{h-1})
```

Runtime: looking for p_0 ("most left point") takes $O(n)$ and h is the number of points that lie on the convex hull's edge $\rightarrow O(h \cdot n)$

LOCAL REPAIR ALGORITHM

1. Sort all the points in P by ascending x coordinate to give $p_1, \dots, p_n$
2. Start with the polygon defined by the border $\langle p_1, \dots, p_{n-1}, p_n, p_{n-1}, p_2 \rangle$
3. Start at p_1 and move towards p_n. Every time we make a right turn (p_{i+2} is right of the line through p_i, p_{i+1}), we remove the middle vertex (p_{i+1}) and check again for possibly newly created right turns starting at the previous vertex (p_{i-1})
4. Once we arrive at p_n, we repeat the same process to go back towards p_2.

LocalRepair$(p_1, p_2, \dots, p_n)$

▷ setzt $(p_1, p_2, \dots, p_n)$, $n \geq 3$, nach x-Koordinate sortiert, voraus

```
1:  q_0 ← p_1
2:  h ← 0
3:  for i ← 2 to n do                          ▷ untere konvexe Hülle, links nach rechts
4:      while h > 0 und q_h links von q_{h-1}p_i do
5:          h ← h − 1
6:      h ← h + 1
7:      q_h ← p_i                  ▷ (q_0, ..., q_h) untere konvexe Hülle von {p_1, ..., p_i}
8:  h' ← h
9:  for i ← n − 1 downto 1 do                  ▷ obere konvexe Hülle, rechts nach links
10:     while h > h' und q_h links von q_{h-1}p_i do
11:         h ← h − 1
12:     h ← h + 1
13:     q_h ← p_i
14: return (q_0, q_1, ..., q_{h-1})  ▷ Ecken der konvexen Hülle, gg. Uhrzeigersinn
```

Runtime: $O(n \cdot \log(n))$ including sorting or $O(n)$ if already sorted.