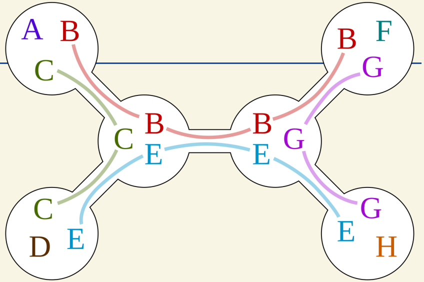
Randomized Methods I: Color Coding



DM898: Parameterized Algorithms
Lars Rohwedder

Today's lecture

- Basics of randomized algorithms
- Longest Path via Color Coding
- Derandomization

Randomized algorithms

A randomized algorithm has access to random bits that it uses to solve a problem

Types of randomized algorithm

- **Las Vegas** algorithm: always correct, but the running time depends on value of random bits
- (one-sided) **Monte Carlo** algorithm (with false negatives): Running time deterministically bounded, always correct when returning YES, sometimes incorrect when returning NO[1]

- In this course, we only consider Monte Carlo algorithms. A useful algorithm should have a **constant** probability $p > 0$, say 99%, of correctly responding on **any** YES-instance
- Randomization allows for elegant and simple algorithmic ideas, which often can be **derandomized**, leading to similar guarantees deterministically
- Most intuitions transfer naturally to randomized algorithms: it is widely believed that $\mathrm{RP} \neq \mathrm{NP}$, i.e., that no Monte Carlo algorithm solves an NP-hard problem with 99% success probability

---

[1] there are also algorithms with two-sided errors or algorithms that are always correct when returning NO and sometimes wrong when returning YES, but in our problems we usually compute a solution which can be checked efficiently. So it is typically easy to avoid falsely returning YES for problems in NP

Boosting probability by repetition

Consider a Monte Carlo algorithm with probability of p of correctly responding in a YES-instance. By repeating the algorithm and outputting NO only if it always returned NO, we can easily boost the probability of correct return value:

The probability of returning the wrong answer after $\lceil 1/p \ln(100) \rceil = O(1/p)$ repetitions is $\leq$

$$(1-p)^{\lceil 1/p \rceil \ln(100)} \leq (e^{-p})^{1/p \cdot \ln(100)} \leq 1/100$$

Bounding by exponential function

For every $x \in \mathbb{R}$: $e^x \geq 1 + x$

Boosting probability by repetition

Consider a Monte Carlo algorithm with probability of p of correctly responding in a YES-instance. By repeating the algorithm and outputting NO only if it always returned NO, we can easily boost the probability of correct return value:

The probability of returning the wrong answer after $\lceil 1/p \ln(100) \rceil = O(1/p)$ repetitions is $\leq$

$$(1-p)^{\lceil 1/p \rceil \ln(100)} \leq (e^{-p})^{1/p \cdot \ln(100)} \leq 1/100$$

Bounding by exponential function

For every $x \in \mathbb{R}$: $e^x \geq 1 + x$

Examples:

- Consider a Monte Carlo algorithm that has a probability of 0.001% of responding correctly in a YES-instance. By increasing the running time with a constant factor, we can make the probability 99%
  $\rightsquigarrow$ **takeaway: precise constant does not matter**
- Consider a Monte Carlo algorithm that has a probability of $1/(2^k n^{10})$ of responding correctly in a YES-instance. By increasing the running time by a factor of $O(2^k n^{10})$ we can make the probability 99%

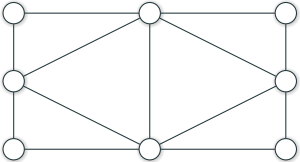
Longest Path

Longest Path

Longest Path problem

Input: Graph $G = (V, E)$, $k \in \mathbb{N}$

Output: YES, if G contains a simple path of length k; NO, otherwise

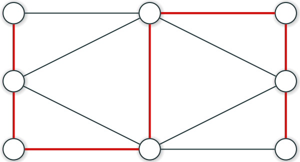


Longest Path

Longest Path problem

Input: Graph $G = (V, E)$, $k \in \mathbb{N}$

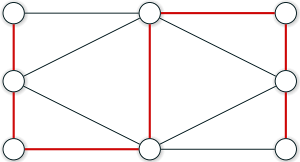
Output: YES, if G contains a simple path of length k; NO, otherwise



Longest Path

Longest Path problem

Input: Graph $G = (V, E)$, $k \in \mathbb{N}$
Output: YES, if G contains a simple path of length k; NO, otherwise



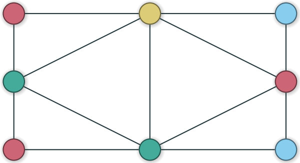
Longest path is NP-hard

Does it have an FPT algorithm in k?

An easier problem

Multicolored Path problem

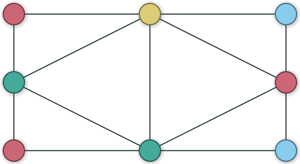
Input: Graph $G = (V, E)$, $k \in \mathbb{N}$, (not necessarily proper) vertex coloring $c : V \to \{1, 2, \ldots, k\}$
Output: YES, if G contains a simple path of length k where each vertex has a different color; NO, otherwise



An easier problem

Multicolored Path problem

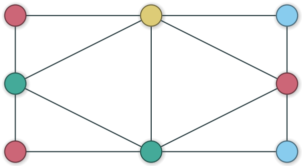
Input: Graph $G = (V, E)$, $k \in \mathbb{N}$, (not necessarily proper) vertex coloring $c : V \to \{1, 2, \dots, k\}$
Output: YES, if G contains a simple path of length k where each vertex has a different color; NO, otherwise



Dynamic program. For each $v \in V$ and $\emptyset \neq S \subseteq \{1, 2, \dots, k\}$ let

$$D[v, S] = \begin{cases} \text{YES} & \text{if there is a path starting in } v \text{ that visits each } C \in S \text{ exactly once, and no other colors} \\ \text{NO} & \text{otherwise} \end{cases}$$

An easier problem



Multicolored Path problem

Input: Graph $G = (V, E)$, $k \in \mathbb{N}$, (not necessarily proper) vertex coloring $c : V \to \{1, 2, \dots, k\}$

Output: YES, if G contains a simple path of length k where each vertex has a different color; NO, otherwise

Dynamic program. For each $v \in V$ and $\emptyset \neq S \subseteq \{1, 2, \dots, k\}$ let

$$D[v, S] = \begin{cases} \text{YES} & \text{if there is a path starting in } v \text{ that visits each } C \in S \text{ exactly once, and no other colors} \\ \text{NO} & \text{otherwise} \end{cases}$$

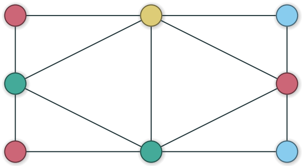
Compute $D[v, S]$ in the order of $|S|$. Base case:

$$D[v, \{C\}] = \begin{cases} \text{YES} & \text{if } C = c(v) \\ \text{NO} & \text{otherwise} \end{cases}$$

An easier problem

Multicolored Path problem

Input: Graph $G = (V, E)$, $k \in \mathbb{N}$, (not necessarily proper) vertex coloring $c : V \to \{1, 2, \ldots, k\}$
Output: YES, if G contains a simple path of length k where each vertex has a different color; NO, otherwise



Dynamic program. For each $v \in V$ and $\emptyset \neq S \subseteq \{1, 2, \ldots, k\}$ let

$$D[v, S] = \begin{cases} \text{YES} & \text{if there is a path starting in } v \text{ that visits each } C \in S \text{ exactly once, and no other colors} \\ \text{NO} & \text{otherwise} \end{cases}$$

Compute $D[v, S]$ in the order of $|S|$. Base case:

$$D[v, \{C\}] = \begin{cases} \text{YES} & \text{if } C = c(v) \\ \text{NO} & \text{otherwise} \end{cases}$$

Recurrence:

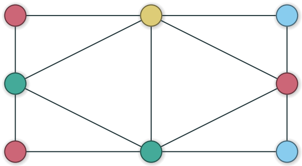
$$D[v, S] = \begin{cases} \bigvee_{u \in N(v)} D[u, S \setminus \{c(v)\}] & \text{if } c(v) \in S \\ \text{NO} & \text{otherwise} \end{cases}$$

An easier problem

Multicolored Path problem

Input: Graph $G = (V, E)$, $k \in \mathbb{N}$, (not necessarily proper) vertex coloring $c : V \to \{1, 2, \ldots, k\}$

Output: YES, if G contains a simple path of length k where each vertex has a different color; NO, otherwise



Dynamic program. For each $v \in V$ and $\emptyset \neq S \subseteq \{1, 2, \ldots, k\}$ let

$$D[v, S] = \begin{cases} \text{YES} & \text{if there is a path starting in } v \text{ that visits each } C \in S \text{ exactly once, and no other colors} \\ \text{NO} & \text{otherwise} \end{cases}$$

Compute $D[v, S]$ in the order of $|S|$. Base case:

$$D[v, \{C\}] = \begin{cases} \text{YES} & \text{if } C = c(v) \\ \text{NO} & \text{otherwise} \end{cases}$$

Recurrence:

$$D[v, S] = \begin{cases} \bigvee_{u \in N(v)} D[u, S \setminus \{c(v)\}] & \text{if } c(v) \in S \\ \text{NO} & \text{otherwise} \end{cases}$$

Running time: $2^k \cdot n^{O(1)}$

Reducing Longest Path to Multicolored Path

Algorithm for Longest Path

- For each $v \in V$ sample $c(v) \in \{1, 2, \ldots, k\}$ uniformly at random
- $\mathrm{out} \leftarrow$ run DP for Multicolored Path instance (G, k, c)
- return out

Reducing Longest Path to Multicolored Path

Algorithm for Longest Path

- For each $v \in V$ sample $c(v) \in \{1, 2, \ldots, k\}$ uniformly at random
- $\mathrm{out} \leftarrow$ run DP for Multicolored Path instance (G, k, c)
- return out

Observation 1: For NO instance, algorithm always returns NO

Observation 2: For YES instance with length k-path P, algorithm returns YES if vertices in P are colored with different colors.

Reducing Longest Path to Multicolored Path

Algorithm for Longest Path

- For each $v \in V$ sample $c(v) \in \{1, 2, \ldots, k\}$ uniformly at random
- $\mathrm{out} \leftarrow$ run DP for Multicolored Path instance (G, k, c)
- return out

Observation 1: For NO instance, algorithm always returns NO

Observation 2: For YES instance with length k-path P, algorithm returns YES if vertices in P are colored with different colors.

Probability of coloring P with different colors. Let $u_1, \ldots, u_k$ be the vertices in P. For every $C_1, \ldots, C_k \in \{1, 2, \ldots, k\}$ we have

$$\mathbb{P}[c(u_1) = C_1 \wedge \cdots \wedge c(u_k) = C_k] = \frac{1}{k^k}$$

Reducing Longest Path to Multicolored Path

Algorithm for Longest Path

- For each $v \in V$ sample $c(v) \in \{1, 2, \ldots, k\}$ uniformly at random
- $\mathrm{out} \leftarrow$ run DP for Multicolored Path instance (G, k, c)
- return out

Observation 1: For NO instance, algorithm always returns NO

Observation 2: For YES instance with length k-path P, algorithm returns YES if vertices in P are colored with different colors.

Probability of coloring P with different colors. Let $u_1, \ldots, u_k$ be the vertices in P. For every $C_1, \ldots, C_k \in \{1, 2, \ldots, k\}$ we have

$$\mathbb{P}[c(u_1) = C_1 \wedge \cdots \wedge c(u_k) = C_k] = \frac{1}{k^k}$$

Out of the k^k coloring of $u_1, \ldots, u_k$ there are $k! \geq (k/e)^k$ colorings that are pairwise different.

Reducing Longest Path to Multicolored Path

Algorithm for Longest Path

- For each $v \in V$ sample $c(v) \in \{1, 2, \ldots, k\}$ uniformly at random
- $\mathrm{out} \leftarrow$ run DP for Multicolored Path instance (G, k, c)
- return out

Observation 1: For NO instance, algorithm always returns NO

Observation 2: For YES instance with length k-path P, algorithm returns YES if vertices in P are colored with different colors.

Probability of coloring P with different colors. Let $u_1, \ldots, u_k$ be the vertices in P. For every $C_1, \ldots, C_k \in \{1, 2, \ldots, k\}$ we have

$$\mathbb{P}[c(u_1) = C_1 \wedge \cdots \wedge c(u_k) = C_k] = \frac{1}{k^k}$$

Out of the k^k coloring of $u_1, \ldots, u_k$ there are $k! \geq (k/e)^k$ colorings that are pairwise different. Thus, the probability that P is colored with different colors is $\geq$

$$\left(\frac{k}{e}\right)^k \frac{1}{k^k} \geq \left(\frac{1}{e}\right)^k \leftarrow \text{success probability}$$

Reducing Longest Path to Multicolored Path

Algorithm for Longest Path

- For each $v \in V$ sample $c(v) \in \{1, 2, \dots, k\}$ uniformly at random
- $\text{out} \leftarrow$ run DP for Multicolored Path instance (G, k, c)
- return out

Observation 1: For NO instance, algorithm always returns NO

Observation 2: For YES instance with length k-path P, algorithm returns YES if vertices in P are colored with different colors.

Probability of coloring P with different colors. Let $u_1, \dots, u_k$ be the vertices in P. For every $C_1, \dots, C_k \in \{1, 2, \dots, k\}$ we have

$$\mathbb{P}[c(u_1) = C_1 \wedge \cdots \wedge c(u_k) = C_k] = \frac{1}{k^k}$$

Out of the k^k coloring of $u_1, \dots, u_k$ there are $k! \geq (k/e)^k$ colorings that are pairwise different. Thus, the probability that P is colored with different colors is $\geq$

$$\left(\frac{k}{e}\right)^k \frac{1}{k^k} \geq \left(\frac{1}{e}\right)^k \leftarrow \textbf{success probability}$$

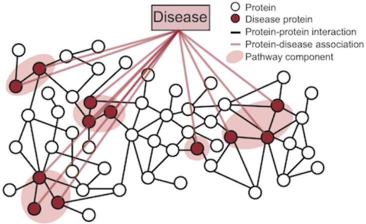
Repeating the algorithm $O(e^k)$ times success probability is constant and the resulting running time is $(2e)^k n^{O(1)}$

Practical Applications

Bioinformatics

Given a graph that represents interactions between proteins in a cell, analyzing the **motifs**, small induced subgraphs, gives valuable insights in biology

Color Coding can be used to obtain statistics about occurrences of specific graphs of size $k \approx 10$ as induced subgraph in large graphs



Source: `https://snap.stanford.edu/pathways/`

Applications here require two extensions:

- finding specific small induced subgraph H (not only paths), known as **subgraph isomorphism**
- counting such subgraphs

Subgraph isomorphism can be solved in FPT time if $\mathrm{tw}(H) = O(1)$. This extends to counting

Derandomization

Family of hash functions

- We want a **perfect** hash function $f : \{1, 2, \dots, n\} \to \{1, 2, \dots, k\}$ such that for an (unknown) $S \subseteq \{1, 2, \dots, n\}$ with $|S| = k$ we have $f(a) \neq f(b)$ for all $a, b \in S, a \neq b$
- It seems impossible to deterministically create such an f without knowing S: for every fixed hash function f there is some S that does not satisfy the above

Family of hash functions

- We want a **perfect** hash function $f : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, k\}$ such that for an (unknown) $S \subseteq \{1, 2, \ldots, n\}$ with $|S| = k$ we have $f(a) \neq f(b)$ for all $a, b \in S, a \neq b$
- It seems impossible to deterministically create such an f without knowing S: for every fixed hash function f there is some S that does not satisfy the above
- **Easier task:** construct family $\mathcal{F}$ of hash functions, such that for each $S \subseteq \{1, 2, \ldots, n\}$ with $|S| = k$ there exists **some** $f \in \mathcal{F}$ with property above
- Then by increasing the running time with a factor of $|\mathcal{F}|$ (and the time to construct the hash functions) we can derandomize Color Coding

Family of hash functions

- We want a **perfect** hash function $f : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, k\}$ such that for an (unknown) $S \subseteq \{1, 2, \ldots, n\}$ with $|S| = k$ we have $f(a) \neq f(b)$ for all $a, b \in S, a \neq b$
- It seems impossible to deterministically create such an f without knowing S: for every fixed hash function f there is some S that does not satisfy the above
- **Easier task:** construct family $\mathcal{F}$ of hash functions, such that for each $S \subseteq \{1, 2, \ldots, n\}$ with $|S| = k$ there exists **some** $f \in \mathcal{F}$ with property above
- Then by increasing the running time with a factor of $|\mathcal{F}|$ (and the time to construct the hash functions) we can derandomize Color Coding

How large does $|\mathcal{F}|$ need to be?

Non-perfect hash function

Consider hash functions of the form $f_q(i) = i \bmod q$ for $q \in \mathbb{N}$

Lemma

Let $S \subseteq \{1, 2, \ldots, n\}$ with $|S| = k$. There exists $q \leq O(k^2 \log n)$ such that $f_q(a) \neq f_q(b)$ for all $a, b \in S, a \neq b$

Non-perfect hash function

Consider hash functions of the form $f_q(i) = i \bmod q$ for $q \in \mathbb{N}$

Lemma

Let $S \subseteq \{1, 2, \ldots, n\}$ with $|S| = k$. There exists $q \leq O(k^2 \log n)$ such that $f_q(a) \neq f_q(b)$ for all $a, b \in S, a \neq b$

Proof. Consider $t = \prod_{a,b \in S, a<b}(b-a) \leq n^{k^2}$

- it is known that $\operatorname{lcm}(\{1, 2, \ldots, m\}) > 2^m$ for $m \geq 7$
- thus, for some $m \leq O(\log t) = O(k^2 \log n)$ we have $\operatorname{lcm}(\{1, 2, \ldots, m\}) > t$
- it follows that there is $q \leq m \leq O(k^2 \log n)$ that does not divide t

Non-perfect hash function

Consider hash functions of the form $f_q(i) = i$ mod q for $q \in \mathbb{N}$

Lemma

Let $S \subseteq \{1, 2, \ldots, n\}$ with $|S| = k$. There exists $q \leq O(k^2 \log n)$ such that $f_q(a) \neq f_q(b)$ for all $a, b \in S, a \neq b$

Proof. Consider $t = \prod_{a,b \in S, a<b} (b - a) \leq n^{k^2}$

- it is known that $\operatorname{lcm}(\{1, 2, \ldots, m\}) > 2^m$ for $m \geq 7$
- thus, for some $m \leq O(\log t) = O(k^2 \log n)$ we have $\operatorname{lcm}(\{1, 2, \ldots, m\}) > t$
- it follows that there is $q \leq m \leq O(k^2 \log n)$ that does not divide t

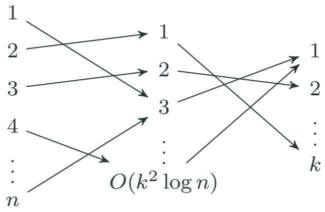
Assume toward constradiction that $f_q(a) = f_q(b)$ for some $a, b \in S$

- Then $(b - a)$ mod $q = 0$. In other words, $b - a$ is a multiple of q
- Thus, q divides t. A contradiction

Constructing a perfect hash function

For each $q \leq O(k^2 \log n)$, $U = \{u_1, u_2, \ldots, u_k\} \subseteq \{1, 2, \ldots, q\}$ define

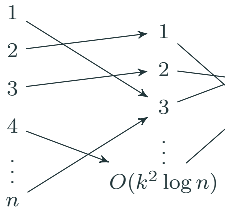
$$f_{q,U}(i) = \begin{cases} 1 & \text{if } f_q(i) = u_1 \\ 2 & \text{if } f_q(i) = u_2 \\ \vdots & \\ k & \text{if } f_q(i) = u_k \\ k & \text{otherwise} \end{cases}$$



Constructing a perfect hash function

For each $q \leq O(k^2 \log n)$, $U = \{u_1, u_2, \ldots, u_k\} \subseteq \{1, 2, \ldots, q\}$ define

$$f_{q,U}(i) = \begin{cases} 1 & \text{if } f_q(i) = u_1 \\ 2 & \text{if } f_q(i) = u_2 \\ \vdots & \\ k & \text{if } f_q(i) = u_k \\ k & \text{otherwise} \end{cases}$$



Let $\mathcal{F} = \{f_{q,U} : q \in \{1, 2, \ldots, m\}, U \subseteq \{1, 2, \ldots, q\}, |U| = k\}$. Then $\mathcal{F}$ contains a perfect hash function for each $S \subseteq \{1, 2, \ldots, n\}$ with $|S| = k$ and

$$|\mathcal{F}| \leq (k^2 \log n)^{k+1} \leq \begin{cases} k^{2(k+1)} \cdot n^{o(1)} & \leq k^{4(k+1)} \cdot n^{o(1)} \text{ if } k \leq \sqrt{\log n} \\ k^{4(k+1)} & \leq k^{4(k+1)} \cdot n^{o(1)} \text{ if } k > \sqrt{\log n} \end{cases}$$

Thus, we can obtain an FPT algorithm for Longest Path and other applications of Color Coding also deterministically

This construction is **not optimized**. There exist more sophisticated hash function families that are much smaller, see e.g. textbook