Coloring Circular Arcs

- Input: We are given an (undirected) cycle with n nodes (and hence n edges), m simple paths of it, and an integer $k > 0$.
- Output: Whether it is possible to color the paths with k colors so that no two paths with the same color share an edge.

Coloring Circular Arcs

- Input: We are given an (undirected) cycle with n nodes (and hence n edges), m simple paths of it, and an integer $k > 0$.
- Output: Whether it is possible to color the paths with k colors so that no two paths with the same color share an edge.
- The problem is NP-complete with a complicated reduction.

Coloring Circular Arcs

- Input: We are given an (undirected) cycle with n nodes (and hence n edges), m simple paths of it, and an integer $k > 0$.
- Output: Whether it is possible to color the paths with k colors so that no two paths with the same color share an edge.
- The problem is NP-complete with a complicated reduction.
- Naïve solution: enumerate all k colorings, running time $O(k^m)$.

Coloring Circular Arcs

- Input: We are given an (undirected) cycle with n nodes (and hence n edges), m simple paths of it, and an integer $k > 0$.
- Output: Whether it is possible to color the paths with k colors so that no two paths with the same color share an edge.
- The problem is NP-complete with a complicated reduction.
- Naïve solution: enumerate all k colorings, running time $O(k^m)$.
- Goal: an algorithm with running time $O(f(k) \cdot \text{poly}(n, m))$, where $f(k)$ is a function of k only. For small values of k this would scale nicely with n and m.

A simpler case: Paths on an interval

- If the graph to start with is not a cycle but a path itself, the problem becomes interval scheduling, and is readily solvable by a greedy algorithm.

A simpler case: Paths on an interval

- If the graph to start with is not a cycle but a path itself, the problem becomes interval scheduling, and is readily solvable by a greedy algorithm.
- If an edge e is shared by $d(e)$ paths, then one needs at least $d(e)$ colors.

A simpler case: Paths on an interval

- If the graph to start with is not a cycle but a path itself, the problem becomes interval scheduling, and is readily solvable by a greedy algorithm.
- If an edge e is shared by $d(e)$ paths, then one needs at least $d(e)$ colors.
- A digression: it turns out that one needs exactly $\max_e d(e)$ colors.

A simpler case: Paths on an interval

- If the graph to start with is not a cycle but a path itself, the problem becomes interval scheduling, and is readily solvable by a greedy algorithm.
- If an edge e is shared by $d(e)$ paths, then one needs at least $d(e)$ colors.
- A digression: it turns out that one needs exactly $\max_e d(e)$ colors.
- This smells of a max-flow min-cut phenomenon. In fact it is indeed a consequence of the max-flow min-cut theorem.

A simpler case: Paths on an interval

- If the graph to start with is not a cycle but a path itself, the problem becomes interval scheduling, and is readily solvable by a greedy algorithm.
- If an edge e is shared by $d(e)$ paths, then one needs at least $d(e)$ colors.
- A digression: it turns out that one needs exactly $\max_e d(e)$ colors.
- This smells of a max-flow min-cut phenomenon. In fact it is indeed a consequence of the max-flow min-cut theorem.

Definition

A *partially ordered* set is a set S equipped with a binary relation $\preceq$ satisfying:

1. *Reflextive*: $\forall a \in S$, $a \preceq a$.
2. *Transitivity*: If $a \preceq b$ and $b \preceq c$, then $a \preceq c$.
3. *Anti-symmetric*: If $a \preceq b$ and $b \preceq a$ then $a = b$.

Examples of partially ordered sets (posets)

If $a \preceq b$ but $b \npreceq a$, we write $a \prec b$.

Examples of partially ordered sets (posets)

If $a \preceq b$ but $b \npreceq a$, we write $a \prec b$.
Examples of partially ordered sets:

- Integers, rationals, reals... (These are *totally ordered sets*)

If $a \preceq b$ but $b \npreceq a$, we write $a \prec b$.
Examples of partially ordered sets:

- Integers, rationals, reals... (These are *totally ordered sets*)
- A set of sets (where $\preceq$ is inclusion $\subseteq$)

Examples of partially ordered sets (posets)

If $a \preceq b$ but $b \not\preceq a$, we write $a \prec b$.
Examples of partially ordered sets:

- Integers, rationals, reals... (These are *totally ordered sets*)
- A set of sets (where $\preceq$ is inclusion $\subseteq$)
- A set of paper boxes, where $\preceq$ is "can be packed in". Formally, let's represent a box by its length, width and height: (a, b, c). Then $(a, b, c) \preceq (a', b', c')$ if

Examples of partially ordered sets (posets)

If $a \preceq b$ but $b \not\preceq a$, we write $a \prec b$.
Examples of partially ordered sets:

- Integers, rationals, reals... (These are *totally ordered sets*)
- A set of sets (where $\preceq$ is inclusion $\subseteq$)
- A set of paper boxes, where $\preceq$ is "can be packed in". Formally, let's represent a box by its length, width and height: (a, b, c). Then $(a, b, c) \preceq (a', b', c')$ if there is a permutation $\sigma : \{a, b, c\} \to \{a, b, c\}$ such that $\sigma(a) \leq a', \sigma(b) \leq b', \sigma(c) \leq c'$.

Examples of partially ordered sets (posets)

If $a \preceq b$ but $b \npreceq a$, we write $a \prec b$.
Examples of partially ordered sets:

- Integers, rationals, reals... (These are *totally ordered sets*)
- A set of sets (where $\preceq$ is inclusion $\subseteq$)
- A set of paper boxes, where $\preceq$ is "can be packed in". Formally, let's represent a box by its length, width and height: (a, b, c). Then $(a, b, c) \preceq (a', b', c')$ if there is a permutation $\sigma : \{a, b, c\} \to \{a, b, c\}$ such that $\sigma(a) \leq a', \sigma(b) \leq b', \sigma(c) \leq c'$.
- Positive integers, where $a \preceq b$ if b can be divded by a.

Definition

A *chain* in a partially ordered set is a set of elements $a_1, \ldots, a_n$ such that $a_1 \prec \ldots \prec a_n$. An *antichain* is a set of elements in a partially ordered set that are mutually uncomparable.

Dilworh's Theorem

Definition

A *chain* in a partially ordered set is a set of elements $a_1, \ldots, a_n$ such that $a_1 \prec \ldots \prec a_n$. An *antichain* is a set of elements in a partially ordered set that are mutually uncomparable.

Theorem (Dilworth's)

The minimum number of disjoint chains needed to cover a partially ordered set is equal to the maximum cardinality of an antichain.

By "cover" we mean every element belongs to one of the chains.

A simpler case: Paths on an interval

- If the graph to start with is not a cycle but a path itself, the problem becomes interval scheduling, and is readily solvable by a greedy algorithm.

A simpler case: Paths on an interval

- If the graph to start with is not a cycle but a path itself, the problem becomes interval scheduling, and is readily solvable by a greedy algorithm.
- If an edge e is shared by $d(e)$ paths, then one needs at least $d(e)$ colors.

A simpler case: Paths on an interval

- If the graph to start with is not a cycle but a path itself, the problem becomes interval scheduling, and is readily solvable by a greedy algorithm.
- If an edge e is shared by $d(e)$ paths, then one needs at least $d(e)$ colors.
- A digression: it turns out that one needs exactly $\max_e d(e)$ colors.
- Exercise: derive the above as a consequence of Dilworth's theorem.

A simpler case: Paths on an interval

- If the graph to start with is not a cycle but a path itself, the problem becomes interval scheduling, and is readily solvable by a greedy algorithm.
- If an edge e is shared by $d(e)$ paths, then one needs at least $d(e)$ colors.
- A digression: it turns out that one needs exactly $\max_e d(e)$ colors.
- Exercise: derive the above as a consequence of Dilworth's theorem.
- However, the problem on the cycle is not as straightforward..

A simpler case: Paths on an interval

- If the graph to start with is not a cycle but a path itself, the problem becomes interval scheduling, and is readily solvable by a greedy algorithm.
- If an edge e is shared by $d(e)$ paths, then one needs at least $d(e)$ colors.
- A digression: it turns out that one needs exactly $\max_e d(e)$ colors.
- Exercise: derive the above as a consequence of Dilworth's theorem.
- However, the problem on the cycle is not as straightforward..
- If for any e, $d(e) > k$, we can return Failure. But then what?

A simpler case: Paths on an interval

- If the graph to start with is not a cycle but a path itself, the problem becomes interval scheduling, and is readily solvable by a greedy algorithm.
- If an edge e is shared by $d(e)$ paths, then one needs at least $d(e)$ colors.
- A digression: it turns out that one needs exactly $\max_e d(e)$ colors.
- Exercise: derive the above as a consequence of Dilworth's theorem.
- However, the problem on the cycle is not as straightforward..
- If for any e, $d(e) > k$, we can return Failure. But then what?
- Let's try enumeration again, a little more cleverly.

A “silly” enumerating algorithm

- Think about the following enumerating algorithm. (We are going to take “one step backward and then two steps forwards”.)

A "silly" enumerating algorithm

- Think about the following enumerating algorithm. (We are going to take "one step backward and then two steps forwards".)
- If a path P_j consists of edges $e_{j_1}, e_{j_2}, \ldots, e_{j_\ell}$, create "segments" $s_{j,1}, \ldots, s_{j,\ell}$. So all the paths now consist of disjoint segments. If a segment s corresponds to an edge e, we say s is *on* e.

A "silly" enumerating algorithm

- Think about the following enumerating algorithm. (We are going to take "one step backward and then two steps forwards".)
- If a path P_j consists of edges $e_{j_1}, e_{j_2}, \ldots, e_{j_\ell}$, create "segments" $s_{j,1}, \ldots, s_{j,\ell}$. So all the paths now consist of disjoint segments. If a segment s corresponds to an edge e, we say s is *on* e.
- Instead of coloring the paths, color the segments: each segment can be colored in k ways. Enumerate all possible colorings. For each coloring the segments, check whether:

A "silly" enumerating algorithm

- Think about the following enumerating algorithm. (We are going to take "one step backward and then two steps forwards".)
- If a path P_j consists of edges $e_{j_1}, e_{j_2}, \ldots, e_{j_\ell}$, create "segments" $s_{j,1}, \ldots, s_{j,\ell}$. So all the paths now consist of disjoint segments. If a segment s corresponds to an edge e, we say s is *on* e.
- Instead of coloring the paths, color the segments: each segment can be colored in k ways. Enumerate all possible colorings. For each coloring the segments, check whether:
  1. All the segments belonging to the same path have the same color.
  2. All the segments on the same edge have different colors.

A "silly" enumerating algorithm

- Think about the following enumerating algorithm. (We are going to take "one step backward and then two steps forwards".)
- If a path P_j consists of edges $e_{j_1}, e_{j_2}, \ldots, e_{j_\ell}$, create "segments" $s_{j,1}, \ldots, s_{j,\ell}$. So all the paths now consist of disjoint segments. If a segment s corresponds to an edge e, we say s is *on* e.
- Instead of coloring the paths, color the segments: each segment can be colored in k ways. Enumerate all possible colorings. For each coloring the segments, check whether:
    1. All the segments belonging to the same path have the same color.
    2. All the segments on the same edge have different colors.
- Running time: $O(k^{\sum_j |P_j|})$.

A "silly" enumerating algorithm

- Think about the following enumerating algorithm. (We are going to take "one step backward and then two steps forwards".)
- If a path P_j consists of edges $e_{j_1}, e_{j_2}, \ldots, e_{j_\ell}$, create "segments" $s_{j,1}, \ldots, s_{j,\ell}$. So all the paths now consist of disjoint segments. If a segment s corresponds to an edge e, we say s is *on* e.
- Instead of coloring the paths, color the segments: each segment can be colored in k ways. Enumerate all possible colorings. For each coloring the segments, check whether:
  1. All the segments belonging to the same path have the same color.
  2. All the segments on the same edge have different colors.
- Running time: $O(k^{\sum_j |P_j|})$.
- Can we improve upon this? Keep in mind: anything with respect to k is cheap; anything with respect to n or m is expensive.

The algorithm

- Observation 1: For each edge, there are at most k segments on it, and there are at most $k!$ possible colorings of them. (Remember, $k!$ is cheap!)

The algorithm

- Observation 1: For each edge, there are at most k segments on it, and there are at most $k!$ possible colorings of them. (Remember, $k!$ is cheap!)
- Let $n(e)$ denote the edge after e clockwise, and $p(e)$ the edge before e clockwise.

The algorithm

- Observation 1: For each edge, there are at most k segments on it, and there are at most $k!$ possible colorings of them. (Remember, $k!$ is cheap!)
- Let $n(e)$ denote the edge after e clockwise, and $p(e)$ the edge before e clockwise.
- Observation 2: Once we fix the coloring Φ of the segments on an edge e, we can enumerate all colorings of the segments on $n(e)$ that are consistent with Φ. (Cheaply!)

- Observation 1: For each edge, there are at most k segments on it, and there are at most $k!$ possible colorings of them. (Remember, $k!$ is cheap!)
- Let $n(e)$ denote the edge after e clockwise, and $p(e)$ the edge before e clockwise.
- Observation 2: Once we fix the coloring Φ of the segments on an edge e, we can enumerate all colorings of the segments on $n(e)$ that are consistent with Φ. (Cheaply!)
- Observation 2′: For a set C of colorings of the segments on an edge e, we can enumerate all colorings of the segments on $n(e)$ that are consistent with *some* coloring in S. (Cheaply!)

The algorithm

- Observation 1: For each edge, there are at most k segments on it, and there are at most $k!$ possible colorings of them. (Remember, $k!$ is cheap!)
- Let $n(e)$ denote the edge after e clockwise, and $p(e)$ the edge before e clockwise.
- Observation 2: Once we fix the coloring Φ of the segments on an edge e, we can enumerate all colorings of the segments on $n(e)$ that are consistent with Φ. (Cheaply!)
- Observation 2′: For a set C of colorings of the segments on an edge e, we can enumerate all colorings of the segments on $n(e)$ that are consistent with *some* coloring in S. (Cheaply!)
- Can we combine the two steps and propogate? Be careful with the circularity of the graph!

The Algorithm

- Observation 3: Combining Observations 2 and 2′, we can "propogate" a coloring:
  - Fix the coloring Φ of the segments on edge e, we can enumerate the set C_1 of all colorings of segments on $n(e)$ that are consistent with Φ; (Cheaply!)
  - Then we can enumerate the set C_2 of all colorings of segments on $n(n(e))$ that is consistent with *some* coloring in C_1, *and also consistent with* Φ; (Cheaply!)

The Algorithm

- Observation 3: Combining Observations 2 and 2′, we can "propogate" a coloring:
  - Fix the coloring Φ of the segments on edge e, we can enumerate the set C_1 of all colorings of segments on $n(e)$ that are consistent with Φ; (Cheaply!)
  - Then we can enumerate the set C_2 of all colorings of segments on $n(n(e))$ that is consistent with *some* coloring in C_1, *and also consistent with* Φ; (Cheaply!)
  - We can go on this procedure: when we have C_i, propogate to C_{i+1} that is all the colorings consistent with Φ and some coloring in C_i.

The Algorithm

- Observation 3: Combining Observations 2 and $2'$, we can “propogate” a coloring:
  - Fix the coloring Φ of the segments on edge e, we can enumerate the set C_1 of all colorings of segments on $n(e)$ that are consistent with Φ; (Cheaply!)
  - Then we can enumerate the set C_2 of all colorings of segments on $n(n(e))$ that is consistent with *some* coloring in C_1, *and also consistent with* Φ; (Cheaply!)
  - We can go on this procedure: when we have C_i, propogate to C_{i+1} that is all the colorings consistent with Φ and some coloring in C_i.
- If we can do this until the edge $p(e)$, we find a valid coloring. If we fail at any step, there is no valid k-coloring.

Analysis

- Running time: each step was "cheap", i.e., the number of steps is only a function of k, and there are n steps. So total running time is $O(f(k)n)$.

Analysis

- Running time: each step was "cheap", i.e., the number of steps is only a function of k, and there are n steps. So total running time is $O(f(k)n)$.
- The key to this algorithm: At each step, when we generate C_{i+1}, we only need the information C_i and Φ (why?).

Analysis

- Running time: each step was "cheap", i.e., the number of steps is only a function of k, and there are n steps. So total running time is $O(f(k)n)$.
- The key to this algorithm: At each step, when we generate C_{i+1}, we only need the information C_i and Φ (why?).
- If we have to enumerate all the intermediate sets between Φ and C_i, the running time will explode.

Analysis

- Running time: each step was "cheap", i.e., the number of steps is only a function of k, and there are n steps. So total running time is $O(f(k)n)$.
- The key to this algorithm: At each step, when we generate C_{i+1}, we only need the information C_i and Φ (why?).
- If we have to enumerate all the intermediate sets between Φ and C_i, the running time will explode.
- This is the essence of dynamic programming: pass only the information necessary for the next step of computation!