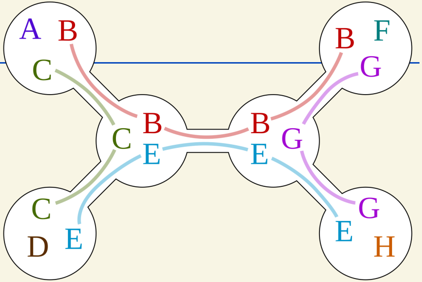
# Branching I

DM898: Parameterized Algorithms
Lars Rohwedder



# Today's lecture

- Formalism of branching
- Vertex Cover
- Closest String

Next lecture: Branch-and-Bound (branching for ILP in practice)

# Formalism of branching

# Branching algorithms for decision problems

By "branching" on some decision, from an instance $I$ we generate several **easier** instances $I_1, \ldots, I_\ell$, $\ell \geq 2$, that we recurse on until we reach trivial instances. Formally, we require that $I_1, \ldots, I_\ell$ satisfy

1. They can be generated from $I$ in polynomial time
2. At least one of $I_1, \ldots, I_\ell$ is YES-instance if and only if $I$ is YES-instance
3. The complexities (or sizes) of $I_1, \ldots, I_\ell$ are each smaller than of $I$ the $\ell$ is bounded.
   **Sufficient for FPT algorithms:** $k(I_1), k(I_2), \ldots, k(I_\ell) < k(I)$ and $\ell = g(k(I))$ for some function $g$

# Branching algorithms for decision problems

By "branching" on some decision, from an instance $I$ we generate several **easier** instances $I_1, \ldots, I_\ell$, $\ell \geq 2$, that we recurse on until we reach trivial instances. Formally, we require that $I_1, \ldots, I_\ell$ satisfy

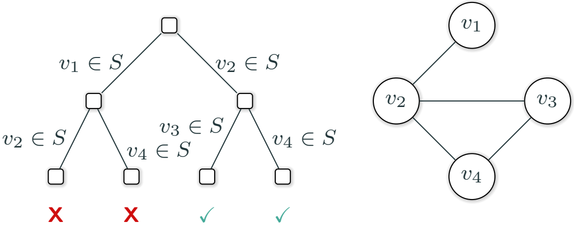
1. They can be generated from $I$ in polynomial time
2. At least one of $I_1, \ldots, I_\ell$ is YES-instance if and only if $I$ is YES-instance
3. The complexities (or sizes) of $I_1, \ldots, I_\ell$ are each smaller than of $I$ the $\ell$ is bounded.

**Sufficient for FPT algorithms:** $k(I_1), k(I_2), \ldots, k(I_\ell) < k(I)$ and $\ell = g(k(I))$ for some function $g$

**Enumeration Tree**

Execution of a branching algorithm is often represented by an **enumeration tree**

- The nodes are the instances, instance is child if generated from other instance
- Edges are labeled with the decisions




Enumeration tree for vertex cover of size 2

# Branching algorithms for decision problems

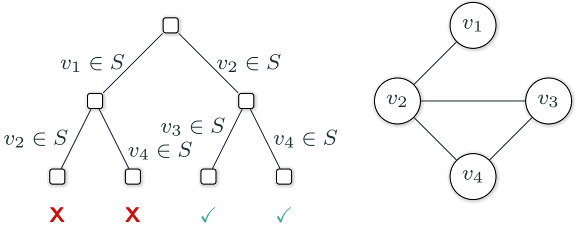
By "branching" on some decision, from an instance $I$ we generate several **easier** instances $I_1, \ldots, I_\ell$, $\ell \geq 2$, that we recurse on until we reach trivial instances. Formally, we require that $I_1, \ldots, I_\ell$ satisfy

1. They can be generated from $I$ in polynomial time
2. At least one of $I_1, \ldots, I_\ell$ is YES-instance if and only if $I$ is YES-instance
3. The complexities (or sizes) of $I_1, \ldots, I_\ell$ are each smaller than of $I$ the $\ell$ is bounded.
   **Sufficient for FPT algorithms:** $k(I_1), k(I_2), \ldots, k(I_\ell) < k(I)$ and $\ell = g(k(I))$ for some function $g$

**Enumeration Tree**

Execution of a branching algorithm is often represented by an **enumeration tree**

- The nodes are the instances, instance is child if generated from other instance
- Edges are labeled with the decisions




Enumeration tree for vertex cover of size 2

Why can this be more efficient than complete enumeration, e.g., of all $\binom{n}{k}$ sets? *Sometimes decisions leading to infeasibility can be identified and ignored early. For example, $v_1, v_2 \notin S$ is never explored in example above.*

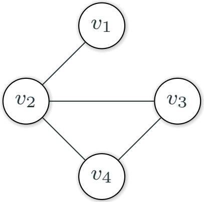
# Vertex Cover revisited

# Branching on vertices

**Branching algorithm**

**Input:** graph $G$, bound $k$ on size of vertex cover.

Choose some $v \in V$. Then



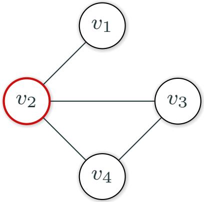
# Branching on vertices

**Branching algorithm**

**Input:** graph $G$, bound $k$ on size of vertex cover.

Choose some $v \in V$. Then

- Either add $v$ to vertex cover, via recursion on $I_1 = (G - v, k - 1)$



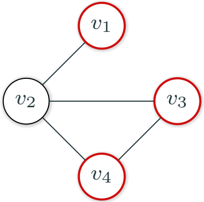
# Branching on vertices

**Branching algorithm**

**Input:** graph $G$, bound $k$ on size of vertex cover.

Choose some $v \in V$. Then

- Either add $v$ to vertex cover, via recursion on $I_1 = (G - v, k - 1)$
- Or add entire neighborhood of $v$ to vertex cover, via recursion on $I_2 = (G - N[v], k - \deg(v))$



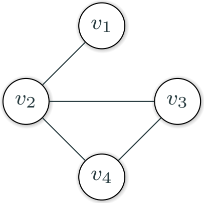
# Branching on vertices

**Branching algorithm**

**Input:** graph $G$, bound $k$ on size of vertex cover.

Choose $v = \operatorname{argmax}_{v \in V} \deg(v)$. Then

- Either add $v$ to vertex cover, via recursion on $I_1 = (G - v, k - 1)$
- Or add entire neighborhood of $v$ to vertex cover, via recursion on $I_2 = (G - N[v], k - \deg(v))$



# Branching on vertices

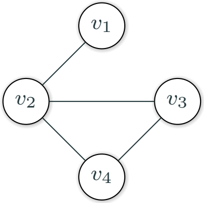
**Branching algorithm**

**Input:** graph $G$, bound $k$ on size of vertex cover.

Choose $v = \text{argmax}_{v \in V} \deg(v)$. Then

- Either add $v$ to vertex cover, via recursion on $I_1 = (G - v, k - 1)$
- Or add entire neighborhood of $v$ to vertex cover, via recursion on $I_2 = (G - N[v], k - \deg(v))$

If maximum degree is $\leq 2$, instance is trivial and we can directly solve it



# Branching on vertices

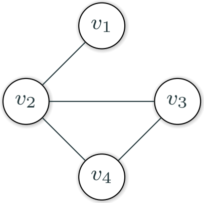
**Branching algorithm**

**Input:** graph $G$, bound $k$ on size of vertex cover.

Choose $v = \mathrm{argmax}_{v \in V} \deg(v)$. Then

- Either add $v$ to vertex cover, via recursion on $I_1 = (G - v, k - 1)$
- Or add entire neighborhood of $v$ to vertex cover, via recursion on $I_2 = (G - N[v], k - \deg(v))$

If maximum degree is $\leq 2$, instance is trivial and we can directly solve it




To analyze the **running time**, bound maximum number of leafs $T(k)$ in enumeration tree:

$$T(k) \leq T(k-1) + T(k-3) \quad \text{if } k \geq 3 \qquad\qquad T(k) = 1 \quad \text{if } k \leq 2$$

# Branching on vertices

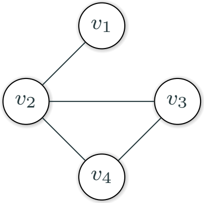
**Branching algorithm**

**Input:** graph $G$, bound $k$ on size of vertex cover.

Choose $v = \text{argmax}_{v \in V} \deg(v)$. Then

- Either add $v$ to vertex cover, via recursion on $I_1 = (G - v, k - 1)$
- Or add entire neighborhood of $v$ to vertex cover, via recursion on $I_2 = (G - N[v], k - \deg(v))$

If maximum degree is $\leq 2$, instance is trivial and we can directly solve it




To analyze the **running time**, bound maximum number of leafs $T(k)$ in enumeration tree:

$$T(k) \leq T(k-1) + T(k-3) \quad \text{if } k \geq 3 \qquad\qquad T(k) = 1 \quad \text{if } k \leq 2$$

**Solving the recurrence**

We want to bound $T(k)$ by $c^k$ for some $c$. It should satisfy:

$$c^{k'} \leq c^{k'-1} + c^{k'-3} \quad \forall k'$$

# Branching on vertices

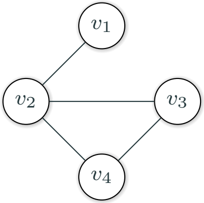
**Branching algorithm**

**Input:** graph $G$, bound $k$ on size of vertex cover.

Choose $v = \text{argmax}_{v \in V} \deg(v)$. Then

- Either add $v$ to vertex cover, via recursion on $I_1 = (G - v, k - 1)$
- Or add entire neighborhood of $v$ to vertex cover, via recursion on $I_2 = (G - N[v], k - \deg(v))$

If maximum degree is $\leq 2$, instance is trivial and we can directly solve it




To analyze the **running time**, bound maximum number of leafs $T(k)$ in enumeration tree:

$$T(k) \leq T(k-1) + T(k-3) \quad \text{if } k \geq 3 \qquad\qquad T(k) = 1 \quad \text{if } k \leq 2$$

**Solving the recurrence**

We want to bound $T(k)$ by $c^k$ for some $c$. It should satisfy:

$$c^{k'} \leq c^{k'-1} + c^{k'-3} \quad \forall k' \quad \Leftrightarrow \quad c^3 \leq c^2 + 1$$

# Branching on vertices

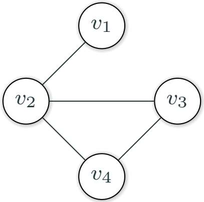
**Branching algorithm**

**Input:** graph $G$, bound $k$ on size of vertex cover.

Choose $v = \mathrm{argmax}_{v \in V} \deg(v)$. Then

- Either add $v$ to vertex cover, via recursion on $I_1 = (G - v, k - 1)$
- Or add entire neighborhood of $v$ to vertex cover, via recursion on $I_2 = (G - N[v], k - \deg(v))$

If maximum degree is $\leq 2$, instance is trivial and we can directly solve it




To analyze the **running time**, bound maximum number of leafs $T(k)$ in enumeration tree:

$$T(k) \leq T(k-1) + T(k-3) \quad \text{if } k \geq 3 \qquad\qquad T(k) = 1 \quad \text{if } k \leq 2$$

**Solving the recurrence**

We want to bound $T(k)$ by $c^k$ for some $c$. It should satisfy:

$$c^{k'} \leq c^{k'-1} + c^{k'-3} \quad \forall k' \quad \Leftrightarrow \quad c^3 \leq c^2 + 1 \quad \Leftarrow \text{(taking } c \text{ as small as possible)} \quad c^3 = c^2 + 1$$

# Branching on vertices

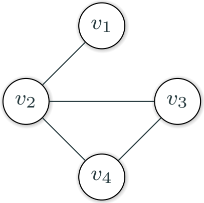
### Branching algorithm

**Input:** graph $G$, bound $k$ on size of vertex cover.

Choose $v = \text{argmax}_{v \in V} \deg(v)$. Then

- Either add $v$ to vertex cover, via recursion on $I_1 = (G - v, k - 1)$
- Or add entire neighborhood of $v$ to vertex cover, via recursion on $I_2 = (G - N[v], k - \deg(v))$

If maximum degree is $\leq 2$, instance is trivial and we can directly solve it




To analyze the **running time**, bound maximum number of leafs $T(k)$ in enumeration tree:

$$T(k) \leq T(k-1) + T(k-3) \quad \text{if } k \geq 3 \qquad\qquad T(k) = 1 \quad \text{if } k \leq 2$$

### Solving the recurrence

We want to bound $T(k)$ by $c^k$ for some $c$. It should satisfy:

$$c^{k'} \leq c^{k'-1} + c^{k'-3} \quad \forall k' \quad \Leftrightarrow \quad c^3 \leq c^2 + 1 \quad \Leftarrow \text{(taking } c \text{ as small as possible)} \quad c^3 = c^2 + 1$$

Using computer tools we can solve the polynomial equation obtaining $c = 1.4656$

Total number of nodes in enumeration tree is at most $2T(k) \rightsquigarrow$ Running time: $T(k) \cdot n^{O(1)} \leq 1.4656^k \cdot n^{O(1)}$

# Closest String

# Problem definition

**Input:** strings $x_1, \ldots, x_k$ of length $L$ over an alphabet $\Sigma$, $d \in \mathbb{Z}_{\geq 0}$

**Output:** decide if there exists string $y$ with $d(x_i, y) \leq d$ for all $i = 1, \ldots, k$

Here, $d(x, y)$ is the **Hamming distance**, the number of characters where the strings differ

# Problem definition

**Input:** strings $x_1, \ldots, x_k$ of length $L$ over an alphabet $\Sigma$, $d \in \mathbb{Z}_{\geq 0}$

**Output:** decide if there exists string $y$ with $d(x_i, y) \leq d$ for all $i = 1, \ldots, k$

Here, $d(x, y)$ is the **Hamming distance**, the number of characters where the strings differ

**Example**

$\Sigma = \{a, b, c\}$

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_1$ | c | a | b | a | c | b | a | b | a | b | a |
| $x_2$ | a | a | b | a | c | b | a | c | a | b | a |
| $x_3$ | a | a | c | a | a | b | a | c | a | b | b |
| $x_4$ | c | b | b | a | a | b | c | c | c | b | a |
| $x_5$ | a | b | b | a | c | b | a | b | a | c | c |
| $y$ | | | | | | | | | | | |

# Problem definition

**Input:** strings $x_1, \ldots, x_k$ of length $L$ over an alphabet $\Sigma$, $d \in \mathbb{Z}_{\geq 0}$

**Output:** decide if there exists string $y$ with $d(x_i, y) \leq d$ for all $i = 1, \ldots, k$

Here, $d(x, y)$ is the **Hamming distance**, the number of characters where the strings differ

**Example**

$\Sigma = \{a, b, c\}$

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_1$ | c | a | b | a | c | b | a | b | a | b | a | |
| $x_2$ | a | a | b | a | c | b | a | c | a | b | a | |
| $x_3$ | a | a | c | a | a | b | a | c | a | b | b | |
| $x_4$ | c | b | b | a | a | b | c | c | c | b | a | |
| $x_5$ | a | b | b | a | c | b | a | b | a | c | c | |
| $y$ | a | a | b | a | c | b | a | c | a | b | a | $\leftarrow$ majority vote |

# Problem definition

**Input:** strings $x_1, \ldots, x_k$ of length $L$ over an alphabet $\Sigma$, $d \in \mathbb{Z}_{\geq 0}$

**Output:** decide if there exists string $y$ with $d(x_i, y) \leq d$ for all $i = 1, \ldots, k$

Here, $d(x, y)$ is the **Hamming distance**, the number of characters where the strings differ

**Example**

$\Sigma = \{a, b, c\}$

| | | | | | | | | | | | | $d(x_i, y)$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_1$ | **c** | a | b | a | c | b | a | **b** | a | b | a | 2 |
| $x_2$ | a | a | b | a | c | b | a | c | a | b | a | 0 |
| $x_3$ | a | a | **c** | a | **a** | b | a | c | a | b | **b** | 3 |
| $x_4$ | **c** | **b** | b | a | **a** | b | **c** | c | **c** | b | a | 5 |
| $x_5$ | a | **b** | b | a | c | b | a | **b** | a | **c** | **c** | 4 |
| $y$ | a | a | b | a | c | b | a | c | a | b | a | |

# Problem definition

**Input:** strings $x_1, \ldots, x_k$ of length $L$ over an alphabet $\Sigma$, $d \in \mathbb{Z}_{\geq 0}$

**Output:** decide if there exists string $y$ with $d(x_i, y) \leq d$ for all $i = 1, \ldots, k$

Here, $d(x, y)$ is the **Hamming distance**, the number of characters where the strings differ

**Example**

$\Sigma = \{a, b, c\}$

| | | | | | | | | | | | | $d(x_i, y)$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_1$ | **c** | **a** | b | a | c | b | a | **b** | a | b | a | 3 |
| $x_2$ | a | **a** | b | a | c | b | a | c | a | b | a | 1 |
| $x_3$ | a | **a** | **c** | a | **a** | b | a | c | a | b | **b** | 4 |
| $x_4$ | **c** | b | b | a | **a** | b | **c** | c | **c** | b | a | 4 |
| $x_5$ | a | b | b | a | c | b | a | **b** | a | **c** | **c** | 3 |
| $y$ | a | **b** | b | a | c | b | a | c | a | b | a | |

# Branching algorithm for Closest String (parameter $d$)

**Idea:** start with some initial solution $y'$ and reduce distance to actual solution $y$ with every branch

Formally each recursive call has input $(y', d', x_1, \ldots, x_k, d)$.
Goal: decide if there exists $y$ with $d(y, y') \leq d'$ and $d(y, x_i) \leq d$ for all $i = 1, \ldots, k$.

| | | | | | | | | | | | | $d(y', x_i)$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_1$ | c | a | b | a | c | b | a | b | a | b | a | 0 |
| $x_2$ | a | a | b | a | c | b | a | c | a | b | a | 2 |
| $x_3$ | a | a | c | a | a | b | a | c | a | b | b | 5 |
| $x_4$ | c | b | b | a | a | b | c | c | c | b | a | 5 |
| $x_5$ | a | b | b | a | c | b | a | b | a | c | c | 4 |
| $y'$ | c | a | b | a | c | b | a | b | a | b | a | |

## Branching algorithm for Closest String (parameter $d$)

**Idea:** start with some initial solution $y'$ and reduce distance to actual solution $y$ with every branch

Formally each recursive call has input $(y', d', x_1, \ldots, x_k, d)$.
Goal: decide if there exists $y$ with $d(y, y') \leq d'$ and $d(y, x_i) \leq d$ for all $i = 1, \ldots, k$.

Start with $(y' = x_1, d' = d, x_1, \ldots, x_d, d)$ **(equivalent to original problem)**

| | | | | | | | | | | | | $d(y', x_i)$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_1$ | c | a | b | a | c | b | a | b | a | b | a | 0 |
| $x_2$ | a | a | b | a | c | b | a | c | a | b | a | 2 |
| $x_3$ | a | a | c | a | a | b | a | c | a | b | b | 5 |
| $x_4$ | c | b | b | a | a | b | c | c | c | b | a | 5 |
| $x_5$ | a | b | b | a | c | b | a | b | a | c | c | 4 |
| $y'$ | c | a | b | a | c | b | a | b | a | b | a | |

## Branching algorithm for Closest String (parameter $d$)

**Idea:** start with some initial solution $y'$ and reduce distance to actual solution $y$ with every branch

Formally each recursive call has input $(y', d', x_1, \ldots, x_k, d)$.
Goal: decide if there exists $y$ with $d(y, y') \leq d'$ and $d(y, x_i) \leq d$ for all $i = 1, \ldots, k$.

Start with $(y' = x_1, d' = d, x_1, \ldots, x_d, d)$ **(equivalent to original problem)**

- If $d(y', x_i) \leq d$ for all $i = 1, \ldots, k$, return YES

| | | | | | | | | | | | | $d(y', x_i)$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_1$ | c | a | b | a | c | b | a | b | a | b | a | 0 |
| $x_2$ | a | a | b | a | c | b | a | c | a | b | a | 2 |
| $x_3$ | a | a | c | a | a | b | a | c | a | b | b | 5 |
| $x_4$ | c | b | b | a | a | b | c | c | c | b | a | 5 |
| $x_5$ | a | b | b | a | c | b | a | b | a | c | c | 4 |
| $y'$ | c | a | b | a | c | b | a | b | a | b | a | |

# Branching algorithm for Closest String (parameter $d$)

**Idea:** start with some initial solution $y'$ and reduce distance to actual solution $y$ with every branch

Formally each recursive call has input $(y', d', x_1, \ldots, x_k, d)$.
Goal: decide if there exists $y$ with $d(y, y') \leq d'$ and $d(y, x_i) \leq d$ for all $i = 1, \ldots, k$.

Start with $(y' = x_1, d' = d, x_1, \ldots, x_d, d)$ **(equivalent to original problem)**

- If $d(y', x_i) \leq d$ for all $i = 1, \ldots, k$, return YES
- Otherwise let $z \in \{x_1, \ldots, x_k\}$ with $d(y', z) > d$. If $d' = 0$ return NO

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | $d(y', x_i)$ | |
| $x_1$ | c | a | b | a | c | b | a | b | a | b | a | 0 | |
| $x_2$ | a | a | b | a | c | b | a | c | a | b | a | 2 | |
| $x_3$ | a | a | c | a | a | b | a | c | a | b | b | 5 | $\leftarrow z$ |
| $x_4$ | c | b | b | a | a | b | c | c | c | b | a | 5 | |
| $x_5$ | a | b | b | a | c | b | a | b | a | c | c | 4 | |
| $y'$ | c | a | b | a | c | b | a | b | a | b | a | | |

# Branching algorithm for Closest String (parameter $d$)

**Idea:** start with some initial solution $y'$ and reduce distance to actual solution $y$ with every branch

Formally each recursive call has input $(y', d', x_1, \ldots, x_k, d)$.
Goal: decide if there exists $y$ with $d(y, y') \leq d'$ and $d(y, x_i) \leq d$ for all $i = 1, \ldots, k$.

Start with $(y' = x_1, d' = d, x_1, \ldots, x_d, d)$ **(equivalent to original problem)**

- If $d(y', x_i) \leq d$ for all $i = 1, \ldots, k$, return YES
- Otherwise let $z \in \{x_1, \ldots, x_k\}$ with $d(y', z) > d$. If $d' = 0$ return NO
- Take any $d + 1$ positions $P$ where $y'$ and $z$ differ. $P$ must contain at least one position $j$ where $y[j] = z[j]$ (if there exists solution $y$)

| | | | | | | | | | | | | $d(y', x_i)$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_1$ | c | a | b | a | c | b | a | b | a | b | a | 0 | |
| $x_2$ | a | a | b | a | c | b | a | c | a | b | a | 2 | |
| $x_3$ | **a** | a | **c** | a | **a** | b | a | **c** | a | b | b | 5 | $\leftarrow z$ |
| $x_4$ | c | b | b | a | a | b | c | c | c | b | a | 5 | |
| $x_5$ | a | b | b | a | c | b | a | b | a | c | c | 4 | |
| $y'$ | **c** | a | **b** | a | **c** | b | a | **b** | a | b | a | | |

# Branching algorithm for Closest String (parameter $d$)

**Idea:** start with some initial solution $y'$ and reduce distance to actual solution $y$ with every branch

Formally each recursive call has input $(y', d', x_1, \ldots, x_k, d)$.
Goal: decide if there exists $y$ with $d(y, y') \leq d'$ and $d(y, x_i) \leq d$ for all $i = 1, \ldots, k$.

Start with $(y' = x_1, d' = d, x_1, \ldots, x_d, d)$ **(equivalent to original problem)**

- If $d(y', x_i) \leq d$ for all $i = 1, \ldots, k$, return YES
- Otherwise let $z \in \{x_1, \ldots, x_k\}$ with $d(y', z) > d$. If $d' = 0$ return NO
- Take any $d + 1$ positions $P$ where $y'$ and $z$ differ. $P$ must contain at least one position $j$ where $y[j] = z[j]$ (if there exists solution $y$)
- Branch over $j \in P$, obtaining instance $(y'', d' - 1, x_1, \ldots, x_k, d)$ where

$$y''[j'] = \begin{cases} y'[j'] & \text{if } j' \neq j \\ z[j] & \text{if } j' = j \end{cases}.$$

| | | | | | | | | | | | | $d(y', x_i)$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_1$ | c | a | b | a | c | b | a | b | a | b | a | 0 | |
| $x_2$ | a | a | b | a | c | b | a | c | a | b | a | 2 | |
| $x_3$ | **a** | a | **c** | a | **a** | b | a | **c** | a | b | b | 5 | $\leftarrow z$ |
| $x_4$ | c | b | b | a | a | b | c | c | c | b | a | 5 | |
| $x_5$ | a | b | b | a | c | b | a | b | a | c | c | 4 | |
| $y'$ | **a** | a | **b** | a | **c** | b | a | **b** | a | b | a | | |

# Branching algorithm for Closest String (parameter $d$)

**Idea:** start with some initial solution $y'$ and reduce distance to actual solution $y$ with every branch

Formally each recursive call has input $(y', d', x_1, \ldots, x_k, d)$.
Goal: decide if there exists $y$ with $d(y, y') \leq d'$ and $d(y, x_i) \leq d$ for all $i = 1, \ldots, k$.

Start with $(y' = x_1, d' = d, x_1, \ldots, x_d, d)$ **(equivalent to original problem)**

- If $d(y', x_i) \leq d$ for all $i = 1, \ldots, k$, return YES
- Otherwise let $z \in \{x_1, \ldots, x_k\}$ with $d(y', z) > d$. If $d' = 0$ return NO
- Take any $d + 1$ positions $P$ where $y'$ and $z$ differ. $P$ must contain at least one position $j$ where $y[j] = z[j]$ (if there exists solution $y$)
- Branch over $j \in P$, obtaining instance $(y'', d' - 1, x_1, \ldots, x_k, d)$ where

$$y''[j'] = \begin{cases} y'[j'] & \text{if } j' \neq j \\ z[j] & \text{if } j' = j \end{cases}.$$

**Running time analysis**

Size of enumeration tree $T(d, d')$:

$$T(d, d') \leq \begin{cases} (d+1) \cdot T(d, d'-1) & \text{if } d' > 0 \\ 1 & \text{if } d' = 0 \end{cases}$$

## Branching algorithm for Closest String (parameter $d$)

**Idea:** start with some initial solution $y'$ and reduce distance to actual solution $y$ with every branch

Formally each recursive call has input $(y', d', x_1, \ldots, x_k, d)$.
Goal: decide if there exists $y$ with $d(y, y') \leq d'$ and $d(y, x_i) \leq d$ for all $i = 1, \ldots, k$.

Start with $(y' = x_1, d' = d, x_1, \ldots, x_d, d)$ **(equivalent to original problem)**

- If $d(y', x_i) \leq d$ for all $i = 1, \ldots, k$, return YES
- Otherwise let $z \in \{x_1, \ldots, x_k\}$ with $d(y', z) > d$. If $d' = 0$ return NO
- Take any $d + 1$ positions $P$ where $y'$ and $z$ differ. $P$ must contain at least one position $j$ where $y[j] = z[j]$ (if there exists solution $y$)
- Branch over $j \in P$, obtaining instance $(y'', d' - 1, x_1, \ldots, x_k, d)$ where

$$y''[j'] = \begin{cases} y'[j'] & \text{if } j' \neq j \\ z[j] & \text{if } j' = j \end{cases}.$$

**Running time analysis**

Size of enumeration tree $T(d, d')$:

$$T(d, d') \leq \begin{cases} (d+1) \cdot T(d, d'-1) & \text{if } d' > 0 \\ 1 & \text{if } d' = 0 \end{cases}$$

$\rightsquigarrow T(d, d') \leq (d+1)^{d'}$

# Branching algorithm for Closest String (parameter $d$)

**Idea:** start with some initial solution $y'$ and reduce distance to actual solution $y$ with every branch

Formally each recursive call has input $(y', d', x_1, \ldots, x_k, d)$.
Goal: decide if there exists $y$ with $d(y, y') \leq d'$ and $d(y, x_i) \leq d$ for all $i = 1, \ldots, k$.

Start with $(y' = x_1, d' = d, x_1, \ldots, x_d, d)$ **(equivalent to original problem)**

- If $d(y', x_i) \leq d$ for all $i = 1, \ldots, k$, return YES
- Otherwise let $z \in \{x_1, \ldots, x_k\}$ with $d(y', z) > d$. If $d' = 0$ return NO
- Take any $d+1$ positions $P$ where $y'$ and $z$ differ. $P$ must contain at least one position $j$ where $y[j] = z[j]$ (if there exists solution $y$)
- Branch over $j \in P$, obtaining instance $(y'', d' - 1, x_1, \ldots, x_k, d)$ where

$$y''[j'] = \begin{cases} y'[j'] & \text{if } j' \neq j \\ z[j] & \text{if } j' = j \end{cases}.$$

**Running time analysis**

Size of enumeration tree $T(d, d')$:

$$T(d, d') \leq \begin{cases} (d+1) \cdot T(d, d'-1) & \text{if } d' > 0 \\ 1 & \text{if } d' = 0 \end{cases}$$

$\rightsquigarrow T(d, d') \leq (d+1)^{d'}$ $\rightsquigarrow$ Running time: $T(d, d) \cdot n^{O(1)} \leq (d+1)^d \cdot n^{O(1)}$