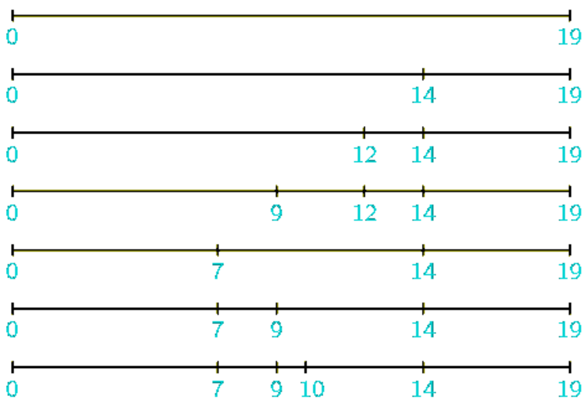
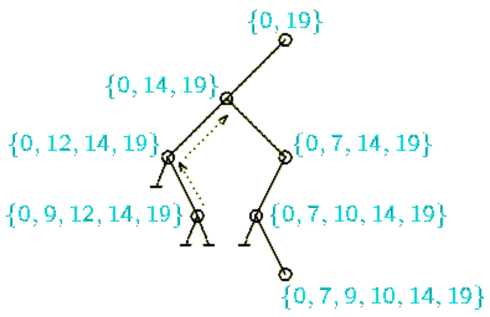
**Example:** $A = \{[1,1],[2,1],[3,1],[4,1],\ [5,2],\ [7,2],[9,2],[10,2],$ $[12,1],[14,1],[19,1]\}$.





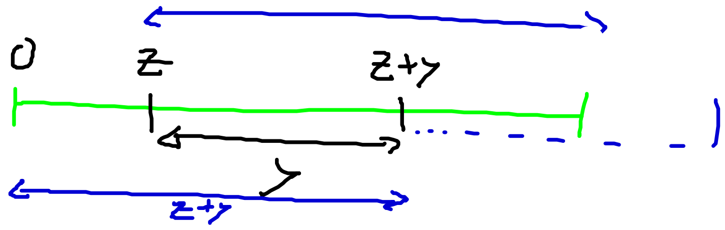
/ dead end, no placement at left (resp. right) end possible.

⇡ Backtracking, saved state is restored.

## Theorem

*If PDP with input $A$ has a feasible solution, our algorithm will find such a solution.*

**Proof:** We show: There are no other solutions than those placing the longest remaining distances at the left or right end.
Consider the partial solution $X = \{0, x_2, x_3, \ldots, y_{max}\}$ constructed so far and denote by $A$ the multiset with all distances implied by $X$ erased. Furthermore let $y$ the largest distance remaining in $A$. Assume $y$ had to be placed in the middle for a correct solution, i.e. with a start point $z \neq 0$ and an end point $z + y \neq y_{max}$.



We distinguish three cases:

1. If $z + y \notin X$, $z + y \in A$ must hold as $0 \in X$ implies distance $|z + y - 0|$. As $z \neq 0$ implies $y < \boxed{z+y}_{\notin A}$ this is a contradiction to our assumption of $y$ being the largest element in $A$.
2. If $z \notin X$, $y_{max} - z \in A$ must hold as $y_{max} \in X$ implies distance $|y_{max} - z|$. As $y > y_{max} - z$ (since $y$ is the largest element of $A$), $y + z > y_{max}$ contradicting $y_{max}$ being right border of the interval considered.
3. If $z \in X$ and $z + y \in X$ we can't place $y$ as planned as this would imply distances of length $0$ not in $A$. $\square$

## Theorem

*For an input with $\binom{n}{2}$ elements the worst case running time of our algorithm is in $\mathcal{O}(2^n \cdot n\log(n))$.*

**Proof:** Initialization possible in time $\mathcal{O}(n^2\log(n))$ (note that $\binom{n}{2} = \frac{n(n-1)}{2}$ holds).

**Worst-Case:** Algorithm tries all possibilities to place the $n-1$ longest fragments at the left or right border.
$\Rightarrow$ Backtracking tree has height $n-1$ (edges) and $\sum_{0\leq i\leq n-1} 2^i = 2^n - 1$ many nodes.
$\Rightarrow$ $2^n - 2$ many edges corresponding to a placement of an element and possibly a backtracking step.
For each placement we have to evaluate $\delta$, possible in running time $\mathcal{O}(n)$ as $|\delta(X, y)|$ in $\mathcal{O}(n)$.
Proving these distances to be in $A$ is done by repeated binary search, resulting in a running time in $\mathcal{O}(n\log(n))$.
By marking chosen distances (instead of deleting them) in $A$ the backtracking step is possible in time $\mathcal{O}(n\log(n))$ analogous to placement.

Adding up all parts we get a total running time in $\mathcal{O}(n^2\log(n) + 2^n \cdot n\log(n)) = \mathcal{O}(2^n \cdot n\log(n))$. □

**Remark:**

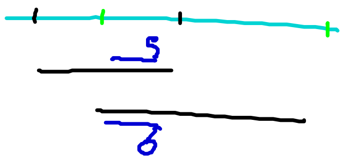
- Worst-Case still exponential but much faster than the brute-force search.
- For real life inputs we have to expect a running time much smaller than the worst-case.
- For a set of $n$ randomly chosen points on the real axis in each step one alternative (left or right placement) is discarded with probability 1 implying a running time in $\mathcal{O}(n^2\log(n))$.

## Mapping with unambiguous probes:

Situation: Many copies of the DNA molecule $D$ to be sequenced are separated into overlapping pieces which are marked unambiguously.

STS probes (sequence-tagged sites): Long parts of DNA sequences, occuring only once in the molecule to be sequenced due to their length. By radioactive markings the STS probes can be distinguished. Hybridization connects the STS probes to the fragments.
$\Rightarrow$ Ideally one gets a 0-1-matrix $A = (a_{i,j})$, where $a_{i,j} = 1$ if and only if STS probe $j$ occurs in the $i$-th fragment.
(Experiments $\Rightarrow$ Errors, forming of chimeras)

Overlapping of fragments $\Rightarrow$ STS probes appear in several fragments.

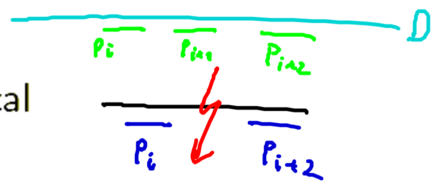


Unambiguousity of probes $\Rightarrow$ Probes mark places of fragments corresponding to the same position in $D$.

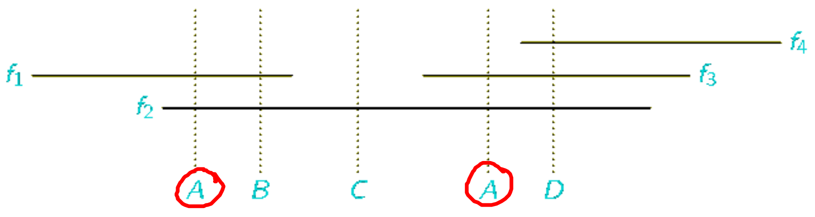
$\Rightarrow$ STS matrix: To reconstruct the molecule the columns have to be permuted so that all ones follow up each other without zeroes in between.

WHY ?

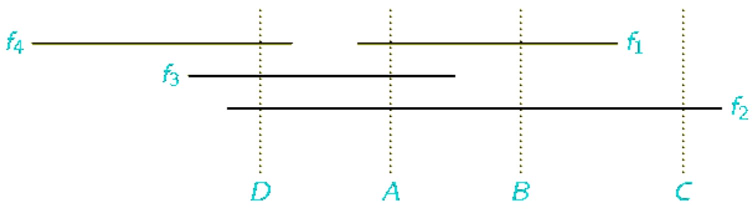
Let $p_1, p_2, \ldots, p_n$ be the correct order of the probes on $D$ and assume there is a fragment containing $p_i$ and $p_{i+2}$ (two ones in one row) but not $p_{i+1}$ (a zero in between). Then $p_{i+1}$ can't be located between $p_i$ and $p_{i+2}$ in $D$, as the fragment is an identical copy of the respective part of $D$.
It is still possible to form a molecule $D'$ from a matrix with non-consecutive ones. In this case, however, $D'$ will contain at least one STS probe twice.



|  | $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|---|
| $f_1$ | 1 | 1 | 0 | 0 |
| $f_2$ | 1 | 1 | 1 | 1 |
| $f_3$ | 1 | 0 | 0 | 1 |
| $f_4$ | 0 | 0 | 0 | 1 |



|  | $D$ | $A$ | $B$ | $C$ |
|---|---|---|---|---|
| $f_1$ | 0 | 1 | 1 | 0 |
| $f_2$ | 1 | 1 | 1 | 1 |
| $f_3$ | 1 | 1 | 0 | 0 |
| $f_4$ | 1 | 0 | 0 | 0 |



## Definition

*Let $A$ be a $n \times m$-matrix over $\{0,1\}$. $A$ has the* consecutive ones property, *if there is a permutation $\pi$ of the columns of $A$, satisfying*

$$(a_{i,\pi^{-1}(k)} = 1 \wedge a_{i,\pi^{-1}(l)} = 1) \longrightarrow (\forall j \in [k+1 : l-1]) : (a_{i,\pi^{-1}(j)} = 1),$$

*for all $1 \leq i \leq n$ and all $1 \leq k < l \leq m$. If this condition holds for $\pi$ the identical permutation, we say, $A$ has* consecutive ones form.

Matrix results from (error free) experiment $\Rightarrow$ consecutive ones property.
Reconstruction of the molecule $\Leftrightarrow$ find permutation $\pi$, arranging the columns of the matrix as needed.

So: Decide if a given matrix has consecutive ones property and if so find a witnessing permutation (consecutive ones problem).

**Brute force:** For realistic $m$ too many alternatives. :-(

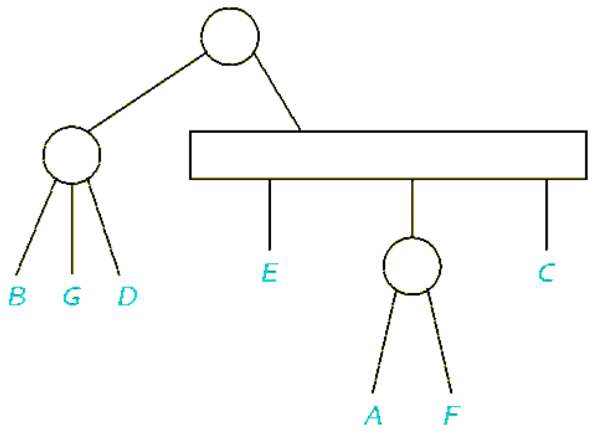
**Solution:** Use a data structure, allowing the efficient maintenance of permutations.

## Definition

*Let $U = \{u_1, \ldots, u_m\}$ a finite set. A* PQ tree $T$ over $U$ *is a 6-tuple $T = (V, E, r, B, l, t)$ satisfying*

1. *$(V, E)$ is an ordered tree,*
2. *$r \in V$ is the root of $(V, E)$,*
3. *$B \subseteq U$ is the set of leaves of $(V, E)$,*
4. *$l : B \to U$ is a **bijective** mapping from the leaves to $U$, and*
5. *$t : V \setminus B \to \{P, Q\}$ is a **total** function, assigning type $P$ or $Q$ to each internal node.*

$\{A, B, C, D, E, F, G\}$, $\bigcirc = P$-node, $\square = Q$-node.



We define $Front(T) := (l(v_1), l(v_2), \ldots, l(v_n))$ for $v_1, v_2, \ldots, v_n$ the sequence of leaves from left to right.

## Definition

*Let $T$ a PQ tree over $U = \{u_1, \ldots, u_m\}$. The following operations on $T$ are* feasible*:*

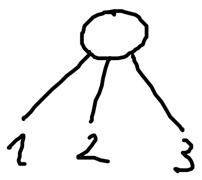
- *Arbitrary variation of the order of children of a $P$-node in $T$;*
- *inverting the order of the children of a $Q$-node in $T$.*

The set $Perm(T)$ of permutations represented by $T$ is then given by

$$Perm(T) := \{Front(T') \mid T \rightsquigarrow T' \text{ by sequence of feasible operations}\}$$

where the empty sequence is allowed as a sequence of feasible operations.

**Example:** Universal PQ tree for $U$ is given by a single $P$-node to which all elements of $U$ are attached as a child.



## Consecutive ones problem:

**Input:** An finite set $U = \{u_1, \ldots, u_m\}$ and a set of restrictions $\mathcal{R} \subseteq 2^U$. (A restriction $R \in \mathcal{R}$ thus is a subset of $U$).
**Output:** All permutations $\pi$ of elements in $U$, such that for all $R \in \mathcal{R}$ the elements of $R$ are successors in $\pi$.

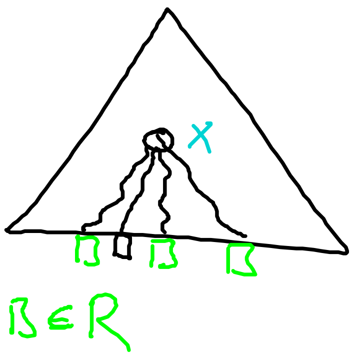
**Method:**
1. Create universal PQ tree for $U$.
2. Subsequently process all restrictions $R$ from $\mathcal{R}$ and transform the PQ tree such that its set $Perm(T)$ satisfies the current and all previous restrictions.

As the satisfaction of an additional restriction generally reduces the size of set $Perm(T)$ we call these transformations *reductions*.

The following implements the reduction of tree $T$ to satisfy restriction $R$:

```
1. $Q :=$ Queue with all leaves of $T$.
2. **while** $Q$ is not empty **do**
    2.1 $x := Dequeue(Q)$;
    2.2 **if** $x$ is a leaf **then**
            **if** $x \in R$ **then**
                mark $x$ as *full*
            **else**
                mark $x$ as *empty*;
    2.3 **else if** a rule can be applied to the subtree with root $x$ **then**
            apply this rule
```

```
2. (Continued)
    2.4 **else**    **return**($\Lambda$) (empty tree, there is no fitting
        permutation);
    2.5 **if** $R \subseteq \{y \mid y$ is a descendant of $x\}$ **then**
            **return**($T$);
    2.6 **if** all siblings of $x$ were considered **then**
```



```
y:=Father(x);
Enqueue(y,Q);
```

Essential are the rules mentioned in 2.3, which we will now consider in detail.

**Notations:** A node $x$ of the PQ tree considered for reduction $R$ is called

- *partial*, if only some of its descendants belong to $R$;
- *empty*, if neither $x$ nor any of its descendants belong to $R$;
- *full*, if $x$ or all of its descendants belong to $R$.

**Rule P0:** If all children of $P$-node $x$ are empty, $T$ remains unchanged.

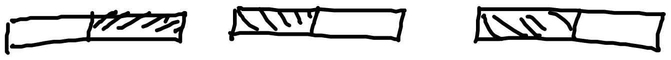
**Rule P1:** If all children of $P$-node $x$ are full, $x$ is marked as full.

**Rule P2:** Only some children of $P$-node $x$ are full and $x$ is the root of the smallest subtree of $T$ containing all of $R$.
Then full and empty children of $x$ are separated by inserting a new (full) $P$-node as child of $x$ getting all full children of $x$ as direct successors.

**Rule P3:** Only some children of $P$-node $x$ are full and $x$ is not the root of the smallest subtree of $T$ containing all of $R$.
Then $x$ is replaced by a partial $Q$-node containing two $P$-nodes as children, one having all full children of $x$ as direct successors, the other having the empty children of $x$ as direct successors.

$P$-nodes with $k$ partial children:
$k = 1$: Rule P4 or P5, depending if $x$ is the root of the smallest subtree containing all of $R$ (P4) or not (P5).
$k = 2$: Rule P6.
$k > 2$: Impossible!

## Why?



For $k > 2$ restriction $R$ can not be satisfied as partial nodes are always $Q$-nodes whose front may only be inverted. So the resulting permutation would contain empty (do not belong to $R$) in between full nodes (belong to $R$). This contradicts our definition of a restriction forcing all elements in $R$ to be neighbored.

In all other possible cases for $P$-nodes full and empty nodes would be nested making satisfaction impossible for the same reasons as above.