Repeat Analysis on a Genomic Scale

Jens Stoye

Genome Informatics, Technology Dept.

and

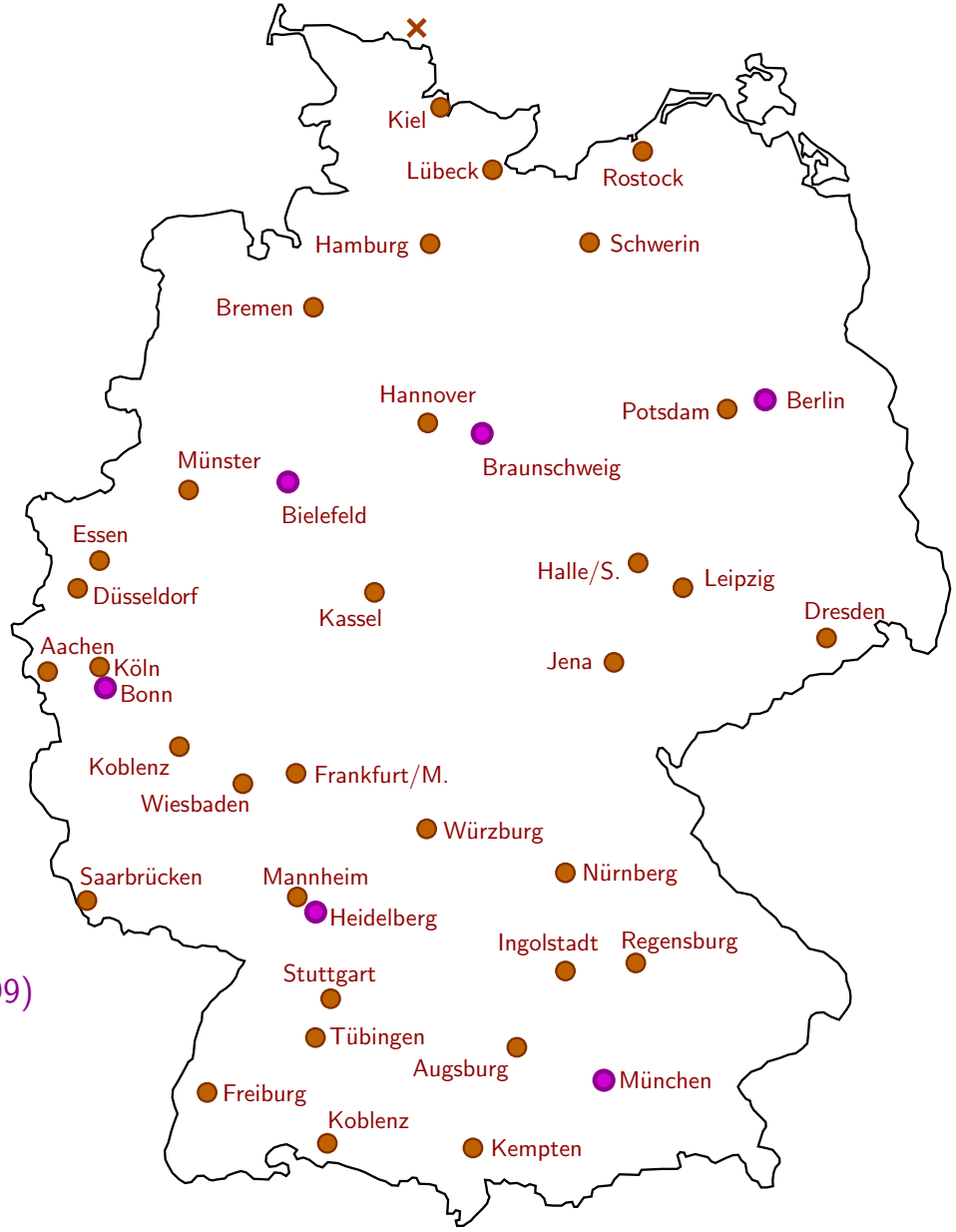
Institute of Bioinformatics, Center of Biotechnology

Bielefeld University, Germany

Bioinformatics in Germany

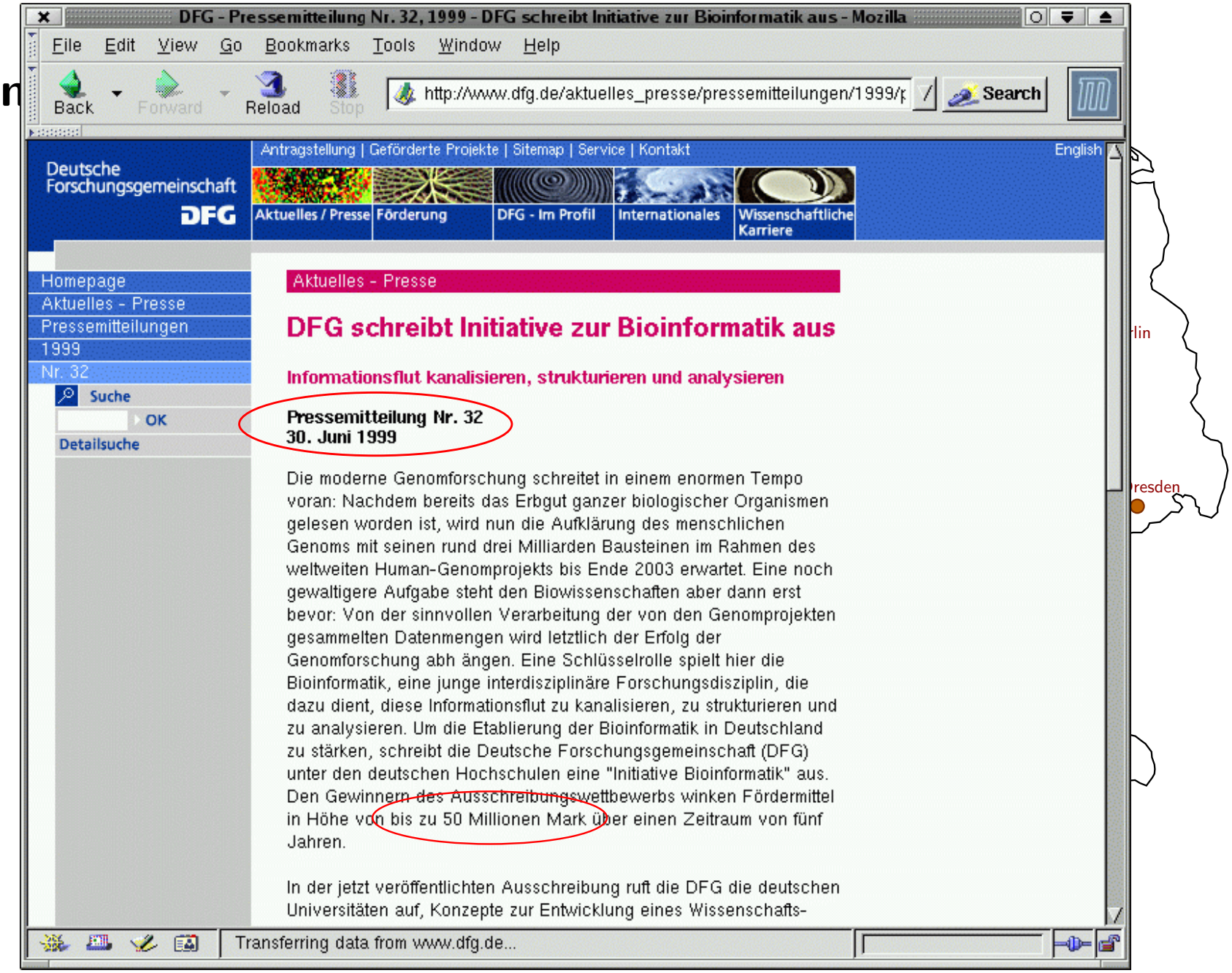
Bioinformatics in Germany

Bioinformatics in Germany

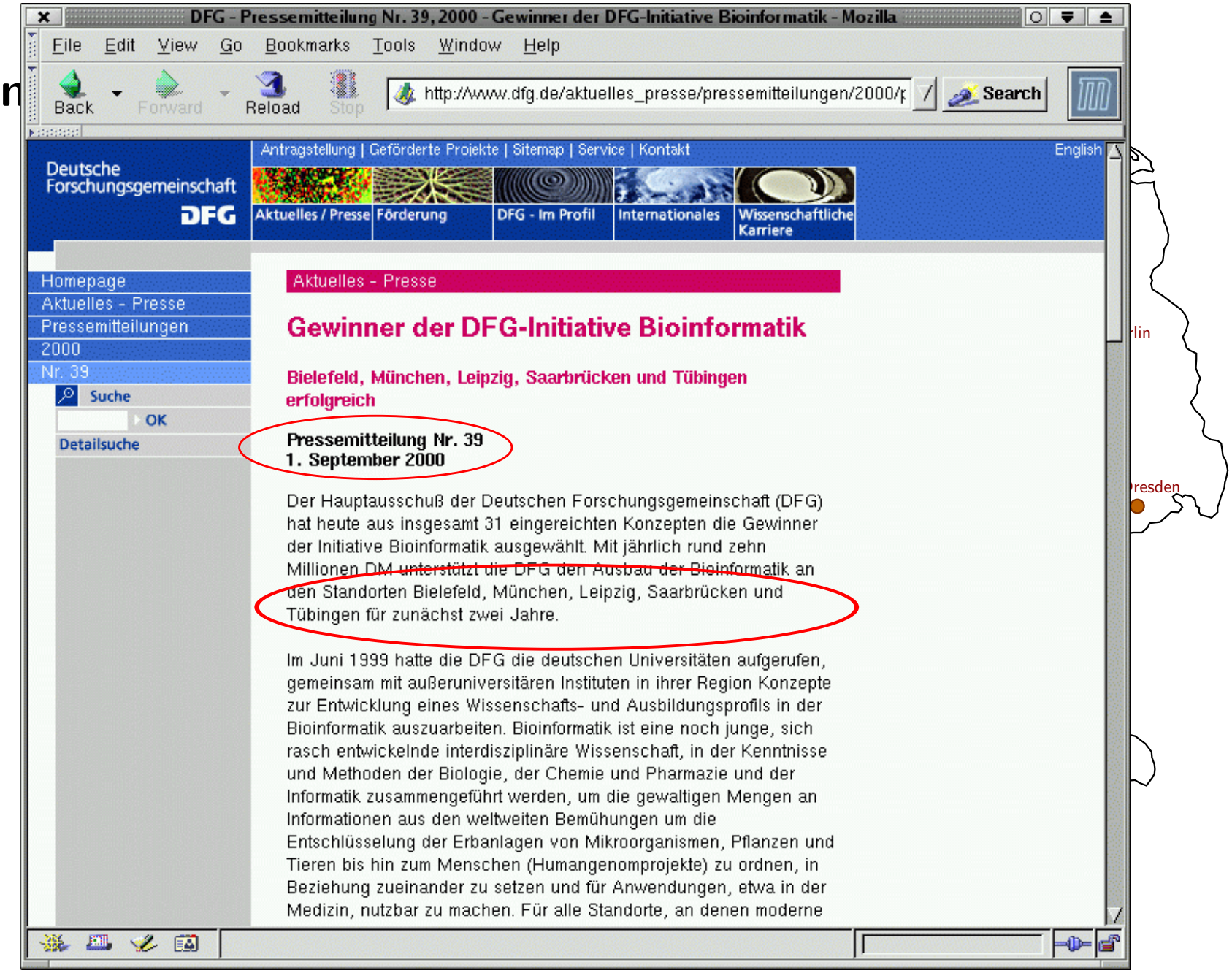


Traditional Bioinformatics places (until ≈ 1999)

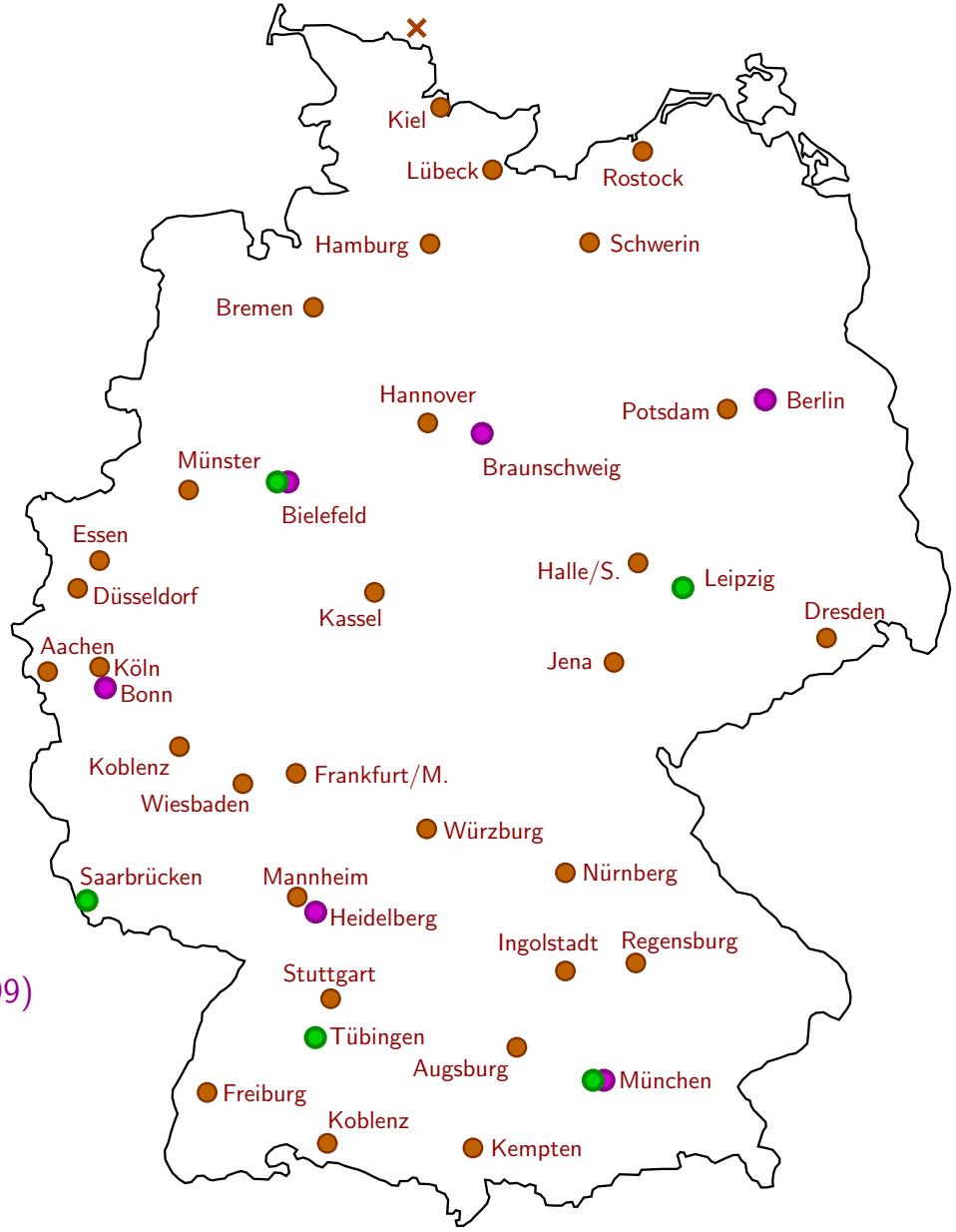
Bioin


DFG - Pressemitteilung Nr. 32, 1999 - DFG schreibt Initiative zur Bioinformatik aus - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://www.dfg.de/aktuelles_presse/pressemitteilungen/1999/
Search
Antragstellung | Geförderte Projekte | Sitemap | Service | Kontakt
English
Deutsche Forschungsgemeinschaft
DFG
Aktuelles / Presse
Förderung
DFG - Im Profil
Internationales
Wissenschaftliche Karriere
Homepage
Aktuelles - Presse
Pressemitteilungen
1999
Nr. 32
Suche
OK
Detailsuche
Aktuelles - Presse
DFG schreibt Initiative zur Bioinformatik aus
Informationsflut kanalisieren, strukturieren und analysieren
Pressemitteilung Nr. 32
30. Juni 1999
Die moderne Genomforschung schreitet in einem enormen Tempo voran: Nachdem bereits das Erbgut ganzer biologischer Organismen gelesen worden ist, wird nun die Aufklärung des menschlichen Genoms mit seinen rund drei Milliarden Bausteinen im Rahmen des weltweiten Human-Genomprojekts bis Ende 2003 erwartet. Eine noch gewaltigere Aufgabe steht den Biowissenschaften aber dann erst bevor: Von der sinnvollen Verarbeitung der von den Genomprojekten gesammelten Datenmengen wird letztlich der Erfolg der Genomforschung abh ängen. Eine Schlüsselrolle spielt hier die Bioinformatik, eine junge interdisziplinäre Forschungsdisziplin, die dazu dient, diese Informationsflut zu kanalisieren, zu strukturieren und zu analysieren. Um die Etablierung der Bioinformatik in Deutschland zu stärken, schreibt die Deutsche Forschungsgemeinschaft (DFG) unter den deutschen Hochschulen eine "Initiative Bioinformatik" aus. Den Gewinnern des Ausschreibungswettbewerbs winken Fördermittel in Höhe von bis zu 50 Millionen Mark über einen Zeitraum von fünf Jahren.
In der jetzt veröffentlichten Ausschreibung ruft die DFG die deutschen Universitäten auf, Konzepte zur Entwicklung eines Wissenschafts-
Transferring data from www.dfg.de...

Bioin


DFG - Pressemitteilung Nr. 39, 2000 - Gewinner der DFG-Initiative Bioinformatik - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://www.dfg.de/aktuelles_presse/pressemitteilungen/2000/
Search
Deutsche Forschungsgemeinschaft DFG
Antragstellung | Geförderte Projekte | Sitemap | Service | Kontakt
English
Aktuelles / Presse
Förderung
DFG - Im Profil
Internationales
Wissenschaftliche Karriere
Homepage
Aktuelles - Presse
Pressemitteilungen
2000
Nr. 39
Suche
OK
Detailsuche
Aktuelles - Presse
Gewinner der DFG-Initiative Bioinformatik
Bielefeld, München, Leipzig, Saarbrücken und Tübingen erfolgreich
Pressemitteilung Nr. 39
1. September 2000
Der Hauptausschuß der Deutschen Forschungsgemeinschaft (DFG) hat heute aus insgesamt 31 eingereichten Konzepten die Gewinner der Initiative Bioinformatik ausgewählt. Mit jährlich rund zehn Millionen DM unterstützt die DFG den Ausbau der Bioinformatik an den Standorten Bielefeld, München, Leipzig, Saarbrücken und Tübingen für zunächst zwei Jahre.
Im Juni 1999 hatte die DFG die deutschen Universitäten aufgerufen, gemeinsam mit außeruniversitären Instituten in ihrer Region Konzepte zur Entwicklung eines Wissenschafts- und Ausbildungsprofils in der Bioinformatik auszuarbeiten. Bioinformatik ist eine noch junge, sich rasch entwickelnde interdisziplinäre Wissenschaft, in der Kenntnisse und Methoden der Biologie, der Chemie und Pharmazie und der Informatik zusammengeführt werden, um die gewaltigen Mengen an Informationen aus den weltweiten Bemühungen um die Entschlüsselung der Erbanlagen von Mikroorganismen, Pflanzen und Tieren bis hin zum Menschen (Humangenomprojekte) zu ordnen, in Beziehung zueinander zu setzen und für Anwendungen, etwa in der Medizin, nutzbar zu machen. Für alle Standorte, an denen moderne

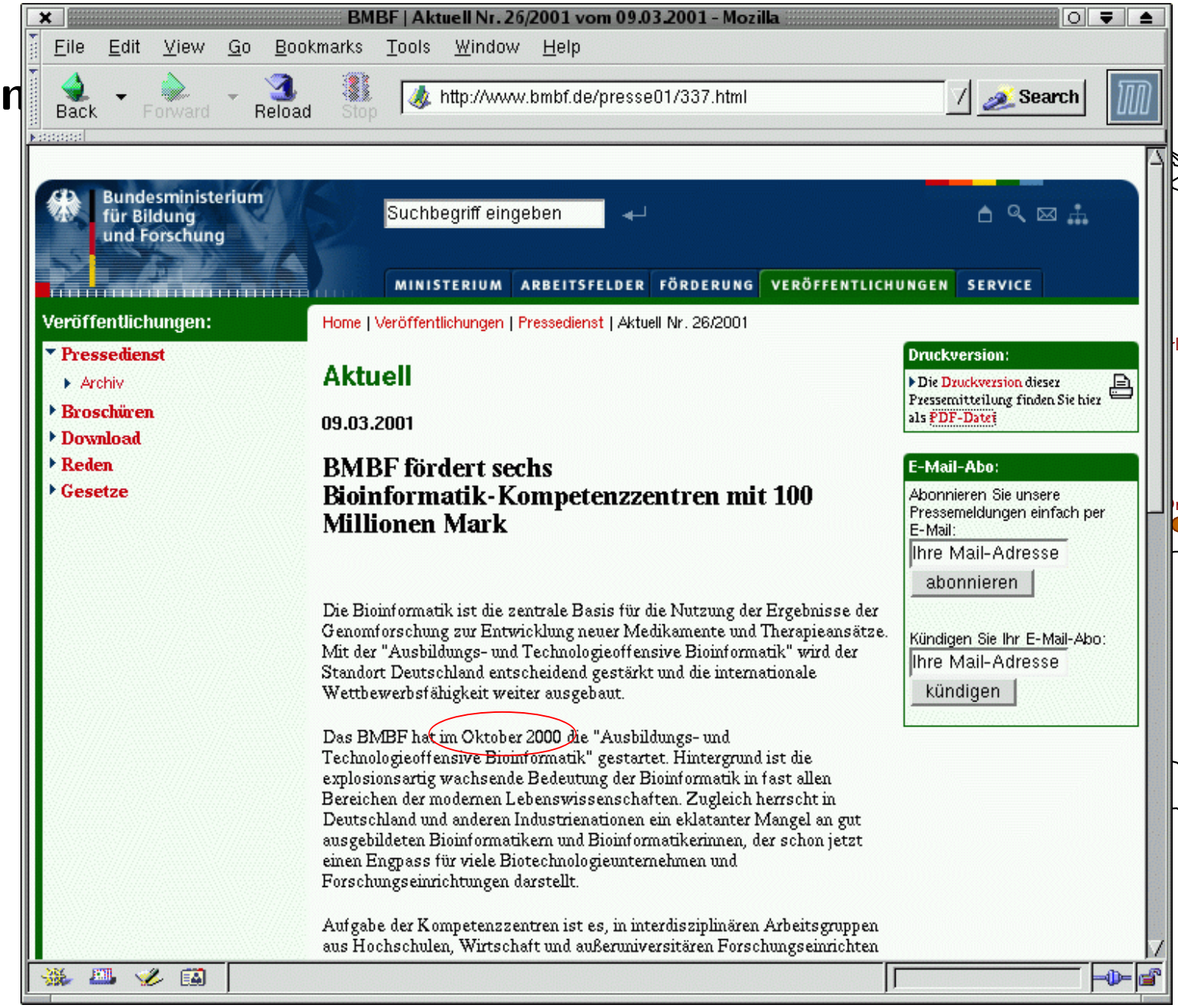
Bioinformatics in Germany



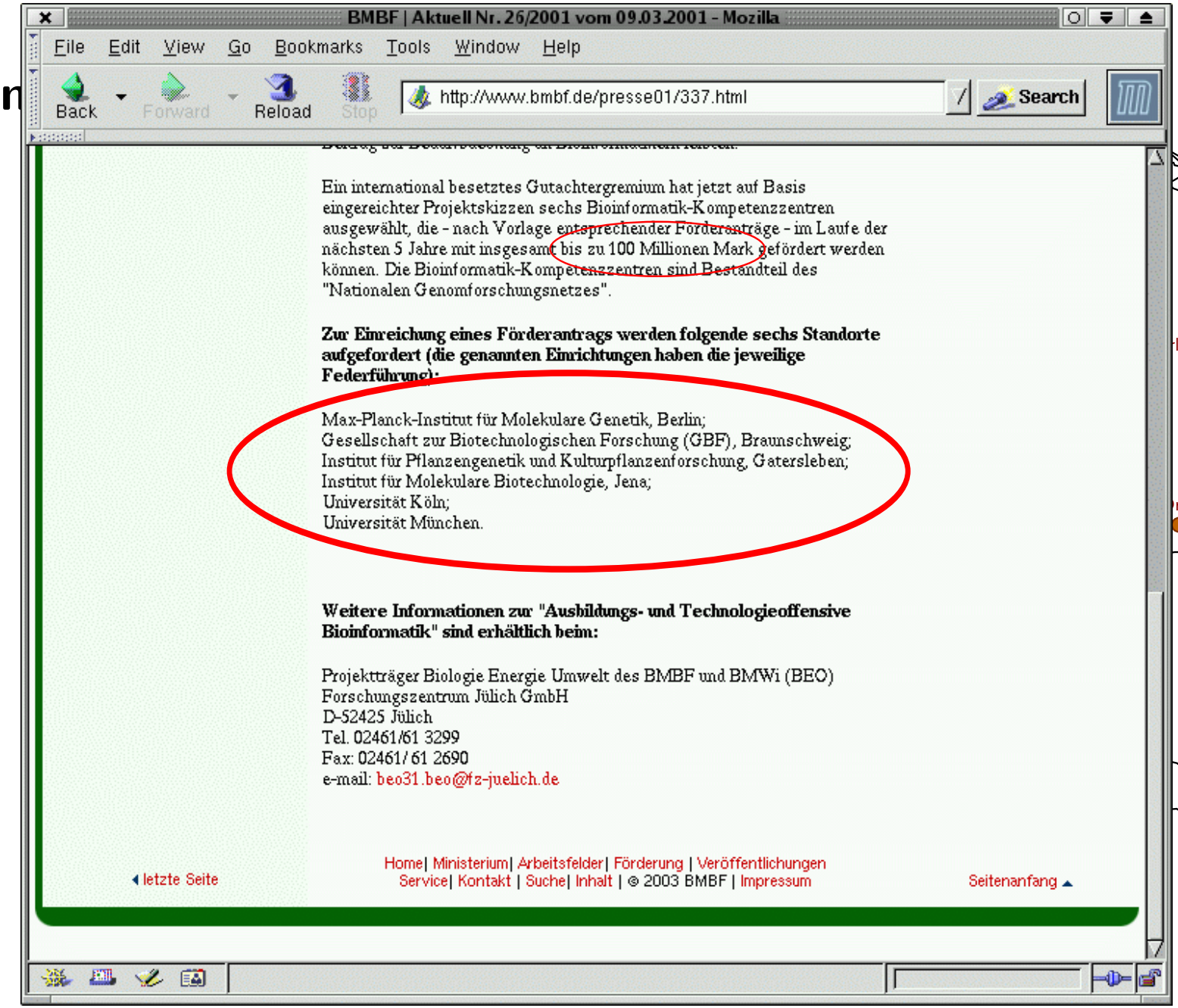
Traditional Bioinformatics places (until ≈ 1999)

DFG-Initiative Bioinformatik (1999)

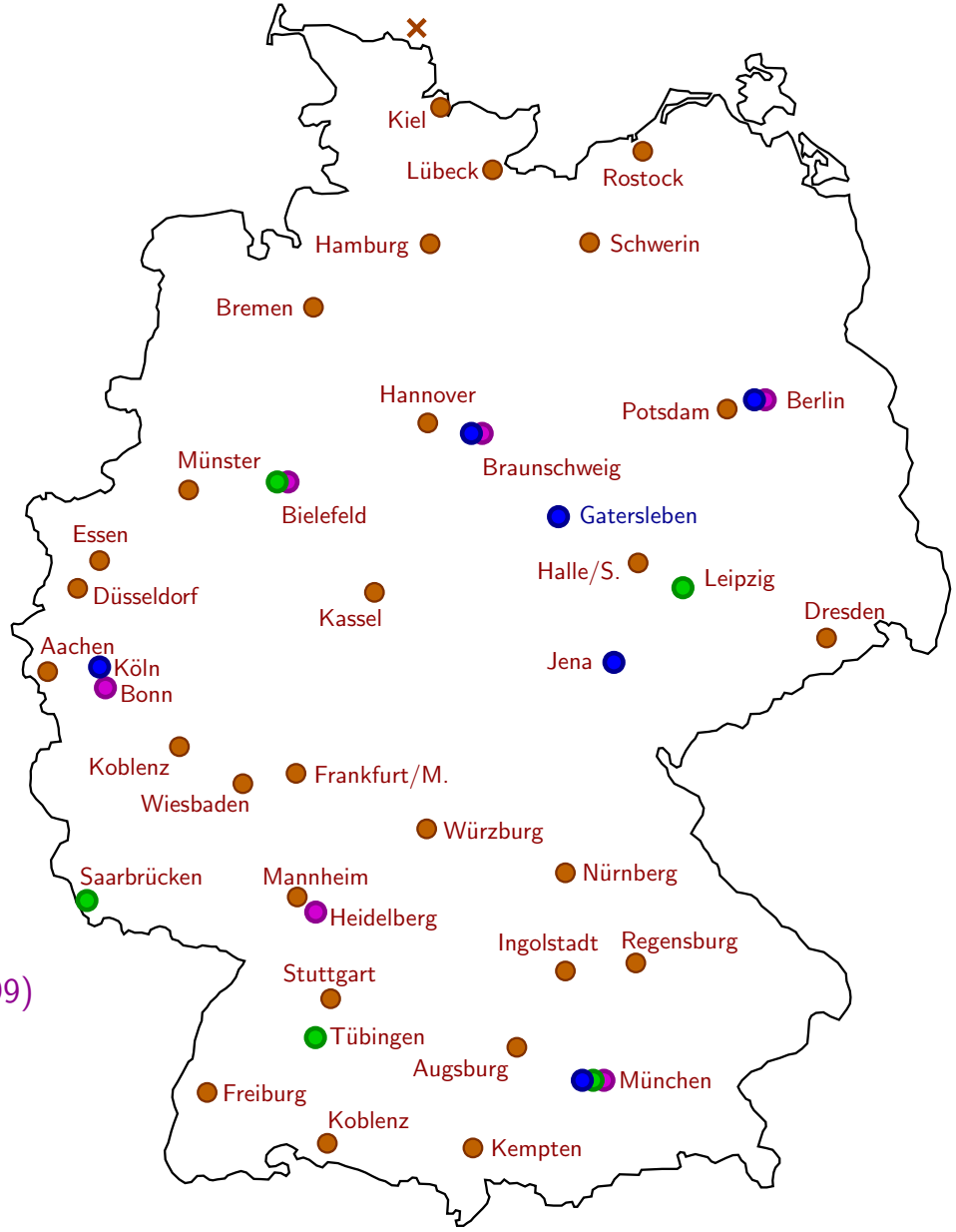
Bioin


BMBF | Aktuell Nr. 26/2001 vom 09.03.2001 - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://www.bmbf.de/presse01/337.html
Search
Bundesministerium für Bildung und Forschung
Suchbegriff eingeben
MINISTERIUM ARBEITSFELDER FÖRDERUNG VERÖFFENTLICHUNGEN SERVICE
Veröffentlichungen:
Pressedienst
Archiv
Broschüren
Download
Reden
Gesetze
Home | Veröffentlichungen | Pressedienst | Aktuell Nr. 26/2001
Aktuell
09.03.2001
BMBF fördert sechs Bioinformatik-Kompetenzzentren mit 100 Millionen Mark
Die Bioinformatik ist die zentrale Basis für die Nutzung der Ergebnisse der Genomforschung zur Entwicklung neuer Medikamente und Therapieansätze. Mit der "Ausbildungs- und Technologieoffensive Bioinformatik" wird der Standort Deutschland entscheidend gestärkt und die internationale Wettbewerbsfähigkeit weiter ausgebaut.
Das BMBF hat im Oktober 2000 die "Ausbildungs- und Technologieoffensive Bioinformatik" gestartet. Hintergrund ist die explosionsartig wachsende Bedeutung der Bioinformatik in fast allen Bereichen der modernen Lebenswissenschaften. Zugleich herrscht in Deutschland und anderen Industrienationen ein eklatanter Mangel an gut ausgebildeten Bioinformatikern und Bioinformatikerinnen, der schon jetzt einen Engpass für viele Biotechnologieunternehmen und Forschungseinrichtungen darstellt.
Aufgabe der Kompetenzzentren ist es, in interdisziplinären Arbeitsgruppen aus Hochschulen, Wirtschaft und außeruniversitären Forschungseinrichten
Druckversion:
Die Druckversion dieser Pressemitteilung finden Sie hier als PDF-Datei
E-Mail-Abo:
Abonnieren Sie unsere Pressemeldungen einfach per E-Mail:
Ihre Mail-Adresse
abonnieren
Kündigen Sie Ihr E-Mail-Abo:
Ihre Mail-Adresse
kündigen

Bioin


BMBF | Aktuell Nr. 26/2001 vom 09.03.2001 - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://www.bmbf.de/presse01/337.html
Search
Ein international besetztes Gutachtergremium hat jetzt auf Basis eingereichter Projektskizzen sechs Bioinformatik-Kompetenzzentren ausgewählt, die - nach Vorlage entsprechender Förderanträge - im Laufe der nächsten 5 Jahre mit insgesamt bis zu 100 Millionen Mark gefördert werden können. Die Bioinformatik-Kompetenzzentren sind Bestandteil des "Nationalen Genomforschungsnetzes".
Zur Einreichung eines Förderantrags werden folgende sechs Standorte aufgefordert (die genannten Einrichtungen haben die jeweilige Federführung):
Max-Planck-Institut für Molekulare Genetik, Berlin;
Gesellschaft zur Biotechnologischen Forschung (GBF), Braunschweig;
Institut für Pflanzengenetik und Kulturpflanzenforschung, Gatersleben;
Institut für Molekulare Biotechnologie, Jena;
Universität Köln;
Universität München.
Weitere Informationen zur "Ausbildungs- und Technologieoffensive Bioinformatik" sind erhältlich beim:
Projektträger Biologie Energie Umwelt des BMBF und BMWi (BEO)
Forschungszentrum Jülich GmbH
D-52425 Jülich
Tel. 02461/61 3299
Fax: 02461/ 61 2690
e-mail: beo31.beo@fz-juelich.de
letzte Seite
Home| Ministerium| Arbeitsfelder| Förderung | Veröffentlichungen
Service| Kontakt | Suche| Inhalt | © 2003 BMBF | Impressum
Seitenanfang

Bioinformatics in Germany



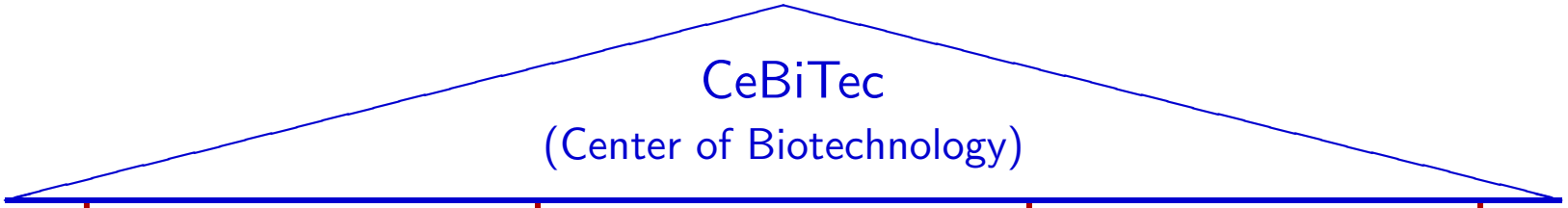
Traditional Bioinformatics places (until ≈ 1999)

DFG-Initiative Bioinformatik (1999)

BMBF Ausbildungs- und Technologieoffensive Bioinformatik (2000)

Bioinformatics in Bielefeld

Bioinformatics in Bielefeld



Institute of Bioinformatics

- Practical Computer Science
- Bio- and Medical Informatics
- Genome Informatics
- Analysis of Mass Spectra

- Bibiserv Tools
- Bibiserv Media

- Theoretical Bioinformatics
- Biomathematics

Graduate School

- 2 scientific groups
- 2 mentors
- 21 PhD grants (up to 3 years each)

Bioinformatics Resource

- Hard- and Software maintenance
- Bioinformatics project support

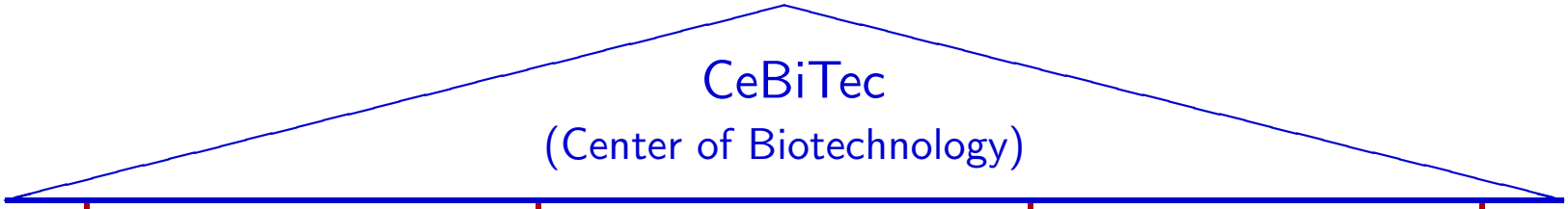
Institute of Genome Research

- Genetics
- Genome Research

- Transcriptomics
- Proteomics

- Central Unit for Genome Research
- Competence Network GenoMik

Bioinformatics in Bielefeld



Institute of Bioinformatics

- Practical Computer Science
- Bio- and Medical Informatics
- Genome Informatics
- Analysis of Mass Spectra
- Bibiserv Tools
- Bibiserv Media
- Theoretical Bioinformatics
- Biomathematics

Graduate School

- 2 scientific groups
- 2 mentors
- 21 PhD grants (up to 3 years each)

Bioinformatics Resource

- Hard- and Software maintenance
- Bioinformatics project support

Institute of Genome Research

- Genetics
- Genome Research
- Transcriptomics
- Proteomics
- Central Unit for Genome Research
- Competence Network GenoMik

Technology Dept.

- Applied Computer Science
- Applied Neuroinformatics
- ...

Biology Dept.

- ...
- ...

Mathematics Dept.

- ...
- ...

Physics Dept.

- ...

Chemistry Dept.

- ...

Bioin

NW-NEWS | Printversion - Mozilla

File Edit View Go Bookmarks Tools Window Help

Back Forward Reload Stop http://www.nw-news.de/cgi-bin/printversion.cgi? Search Print

Home Bookmarks DB Entrez-PubMed Google LEO

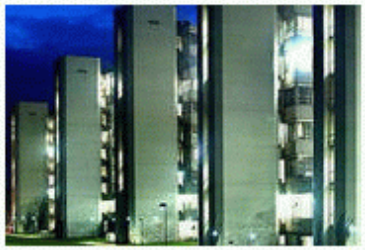
Neue Westfälische im Internet
nw-news.de

LOKALARTIKEL

29.04. 2003

Die Uni wächst weiter

Neues Laborgebäude für Naturwissenschaften / Geplanter Baubeginn 2004



Bielefeld. Die Universität Bielefeld bekommt mehr Platz. Im Nordwesten der Hochschule soll ein neues Laborgebäude für Genomforschung, Biophysik und -informatik entstehen. Baubeginn könnte nach Informationen der Redaktion des Haller Kreisblatts im Frühjahr 2004 sein.

Standort des 44 Meter langen und 20 Meter breiten fünfgeschossigen Gebäudes ist ein Teil des Parkplatzes neben den Sportanlagen im Nordwesten (Wellensiek). Zunächst sollen Labore und Büros mit einer Nutzfläche von 2.100 Quadratmetern gebaut werden, später könnte die Fläche auf 3.300 Quadratmeter vergrößert werden. Bauvolumen der ersten Bauphase: zwölf Millionen Euro. Die Planung liegt beim Bau- und Liegenschaftsbetrieb des Landes Nordrhein-Westfalen.

Diese Erweiterung sei "dringend notwendig", sagt Hartmut Krauß, Planungsdezernent der Universität. Für die seit zehn Jahren wachsende Technische Fakultät sind Labors mit höchstem Standard geplant. Der Neubau ist eine Reaktion auf die Raumnot der aufstrebenden Naturwissenschaften. Das Land Nordrhein-Westfalen hat das Bielefelder Projekt ganz oben auf seine Prioritätenliste gesetzt.

Schwerpunkte der Biologen im neuen Erweiterungsbau werden die mikrobiologische und Pflanzen-Genomforschung sein. Die Biophysik wird vor allem Nanoforschung betreiben. Für ultraempfindliche Mikroskope etwa müssen besondere Raumbedingungen geschaffen werden.

Mit dem Hauptgebäude der 30 Jahre jungen Universität, den so genannten Zähnen, wird der Neubau zunächst nicht verbunden sein, wie Uni-Kanzler Hans-Jürgen Simm erklärt. Für den zweiten Bauabschnitt sei diese Verbindung im Gespräch.

VON ELMAR KRAMER

BILD: Die Uni-Zähne bei Nacht: Läuft alles nach Plan, bekommen sie im nächsten Jahr Zuwachs. FOTO: ANDREAS FRÜCHT

Transferring data from www.nw-news.de...

Overview: Repeat analysis on a genomic scale

- Introduction
  - String pattern matching
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats
  - Degenerate repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

Overview: Repeat analysis on a genomic scale

- Introduction
  - String pattern matching
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats
  - Degenerate repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

Overview: Repeat analysis on a genomic scale

- Introduction
  - String pattern matching
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats
  - Degenerate repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

Pattern matching in biological sequence analysis

Finding *known* patterns: (exact/approximate)

- Search for homologous proteins
  - assumption: similar sequence $\longrightarrow$ similar structure $\longrightarrow$ similar function
- Search for given sequence or structural pattern
  - mapping of *expressed sequence tags* (ESTs) on genomic DNA
  - palindromic or other RNA structural patterns
- *Known repeats* or *low complexity regions* (for further exclusion from analysis)

Pattern matching in biological sequence analysis

Finding *known* patterns: (exact/approximate)

- Search for homologous proteins
  - assumption: similar sequence $\longrightarrow$ similar structure $\longrightarrow$ similar function
- Search for given sequence or structural pattern
  - mapping of *expressed sequence tags* (ESTs) on genomic DNA
  - palindromic or other RNA structural patterns
- *Known repeats* or *low complexity regions* (for further exclusion from analysis)

Finding *structural* patterns: (exact/approximate)

- *Ab initio* gene prediction (start/stop codons, exons/introns)
- Search for over-/underrepresented substrings/-sequences, for example
  - unknown promoter binding sites
  - repeats, tandem repeats
  - possible DNA methylation sites
- Calculation of RNA secondary structure

Requirements for computational tools

- Efficiency: linear in space and time
- Flexibility: applicable to as many problems as possible
- Statistical assessment of significance of results
- Visualization

Requirements for computational tools

- Efficiency: linear in space and time
- Flexibility: applicable to as many problems as possible
- Statistical assessment of significance of results
- Visualization

For routine tasks, even linear time is intolerable $\rightarrow$ index

Many indices for massive sequence data use the property that the text is partitioned into words (e.g. natural language, syntactic tags).

Genomic data is not divided into obvious "words".

We need an index that allows access to any substring of the text.

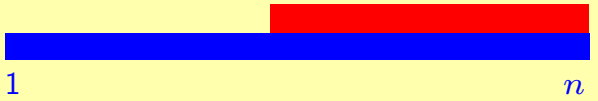
$\rightarrow$ Suffix Tree

Overview: Repeat analysis on a genomic scale

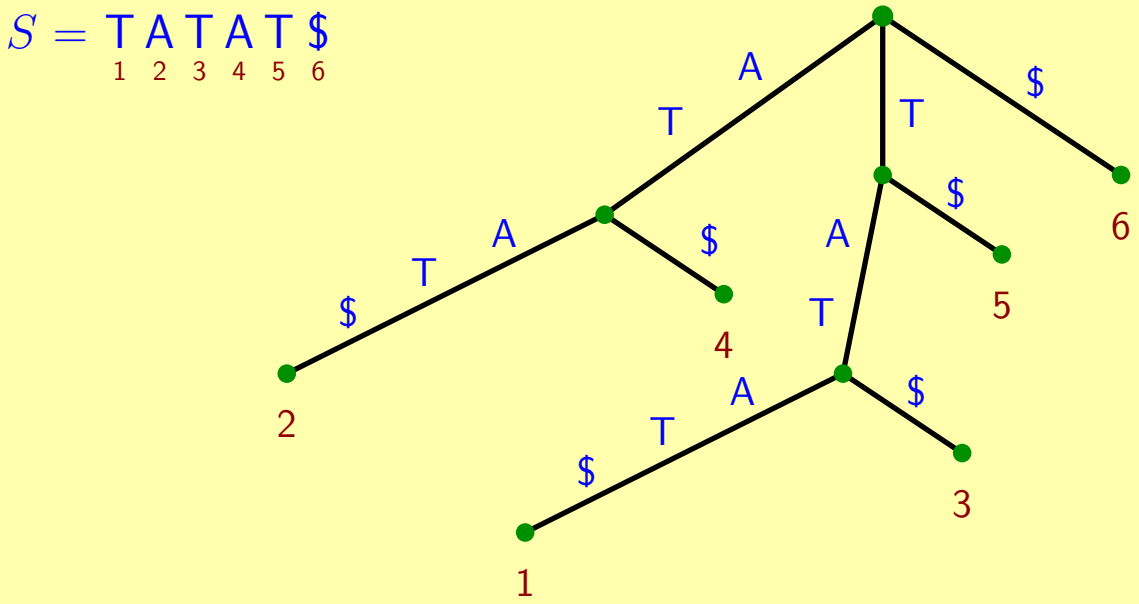
- Introduction
  - String pattern matching
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats
  - Degenerate repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

Suffix Tree: Definition

- A suffix of a string S of length n
  is a substring of S that ends at position n.

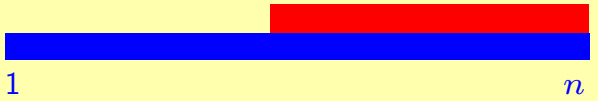


- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.



Suffix Tree: Definition

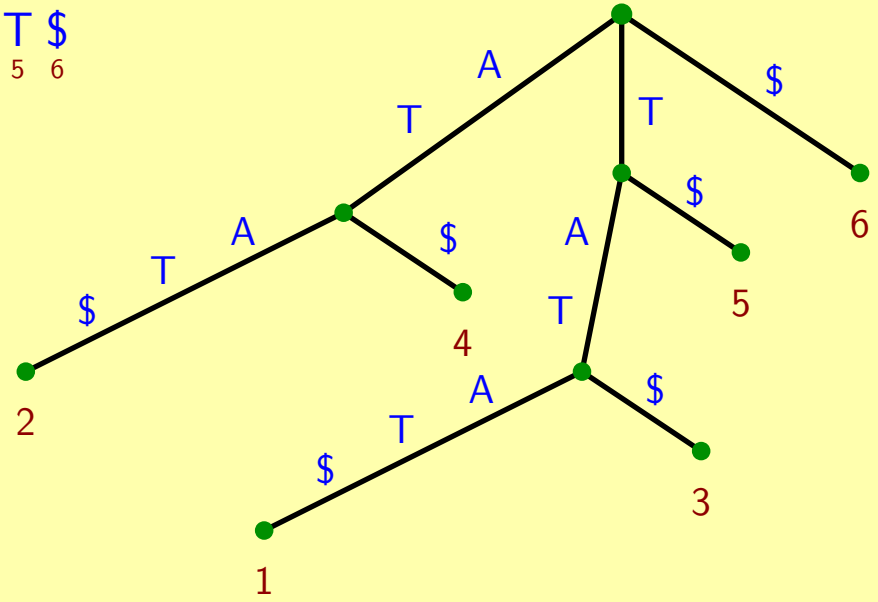
- A suffix of a string S of length n
  is a substring of S that ends at position n.



- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.

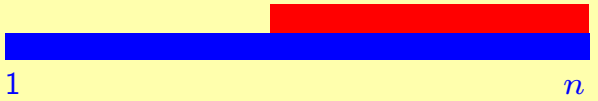
$S =$ T A T A T $
1 2 3 4 5 6

$P =$ A T A



Suffix Tree: Definition

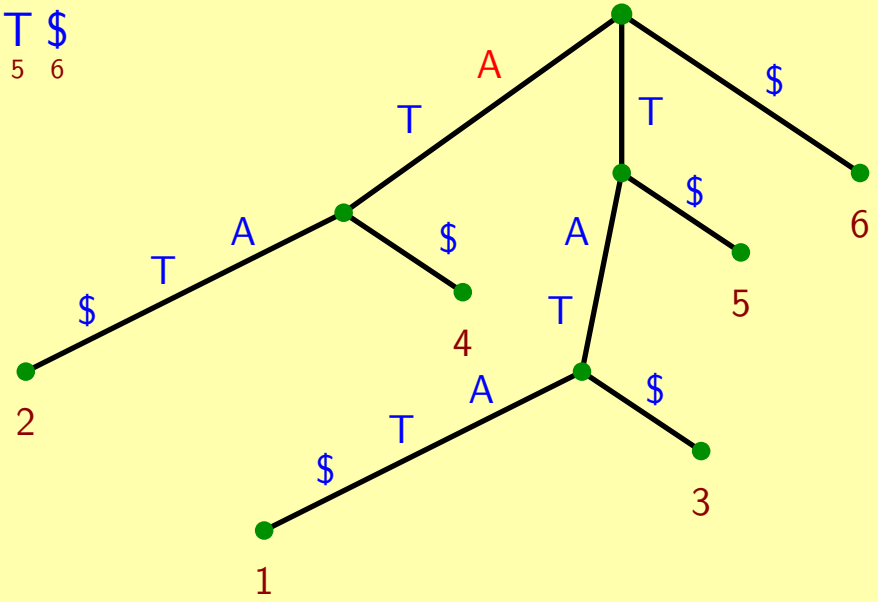
- A suffix of a string S of length n
  is a substring of S that ends at position n.



- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.

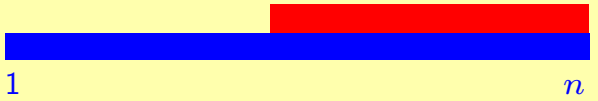
$S =$ T A T A T $
1 2 3 4 5 6

$P =$ A T A



Suffix Tree: Definition

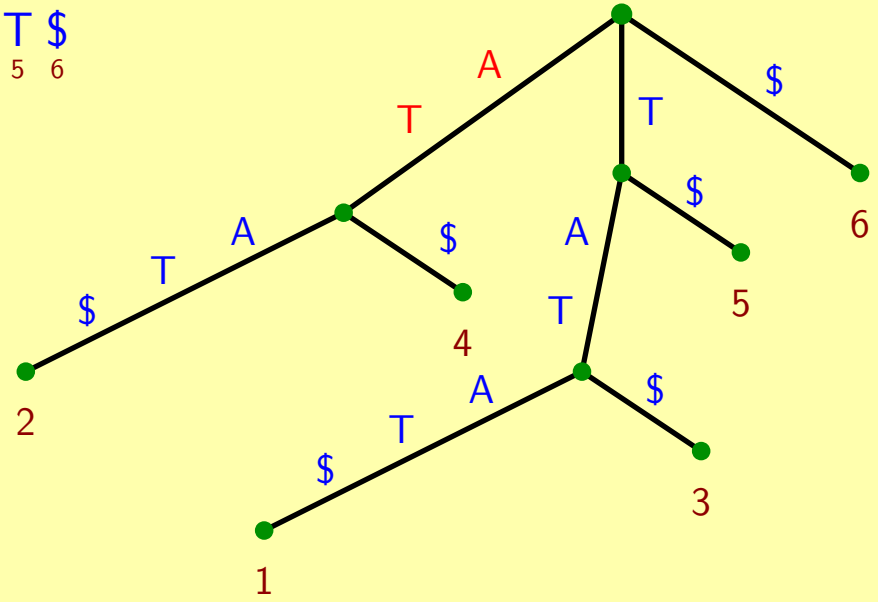
- A suffix of a string S of length n
  is a substring of S that ends at position n.



- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.

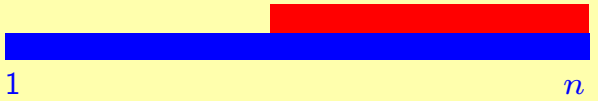
$S =$ T A T A T $
1 2 3 4 5 6

$P =$ A T A



Suffix Tree: Definition

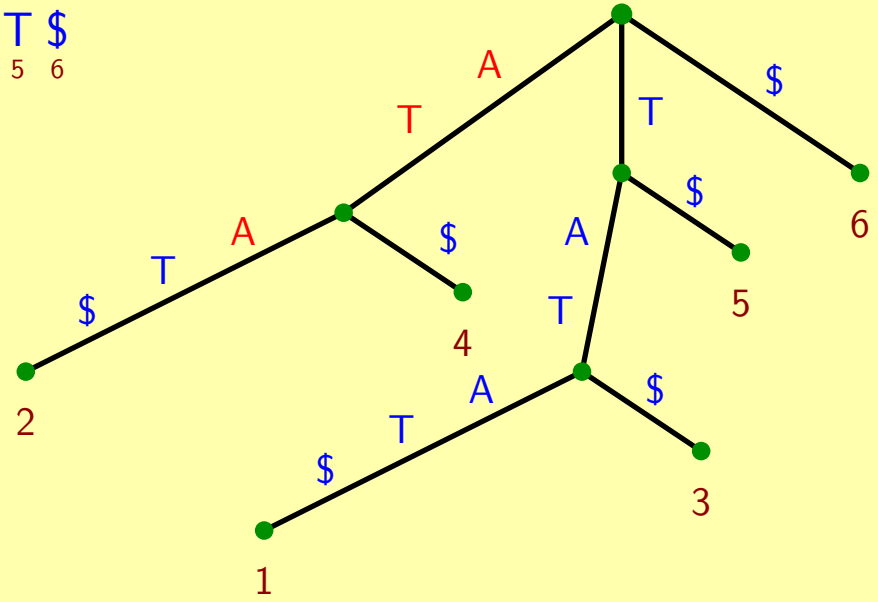
- A suffix of a string S of length n
  is a substring of S that ends at position n.



- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.

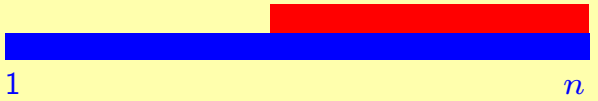
$S =$ T A T A T $
1 2 3 4 5 6

$P =$ A T A



Suffix Tree: Definition

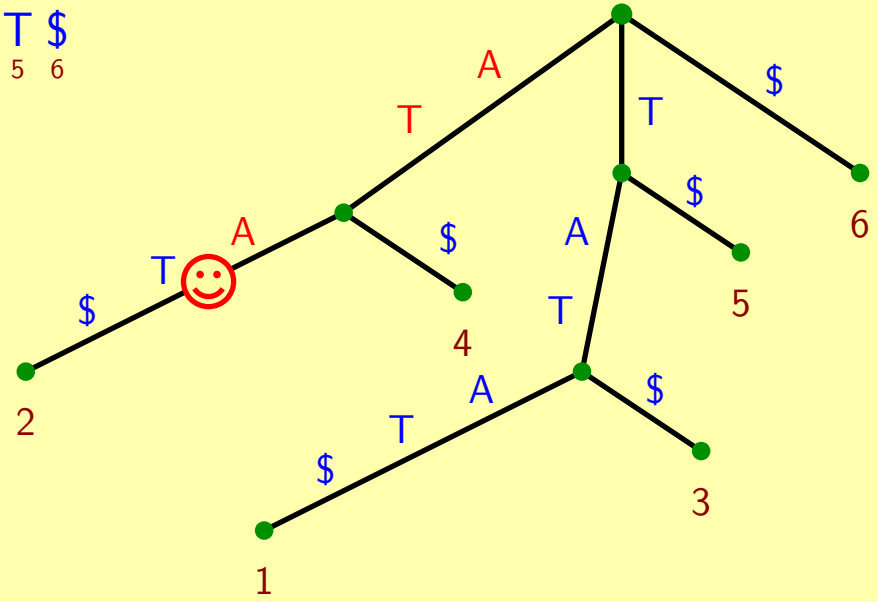
- A suffix of a string S of length n
  is a substring of S that ends at position n.



- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.

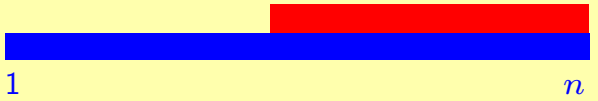
$S =$ T A T A T $
1 2 3 4 5 6

$P =$ A T A

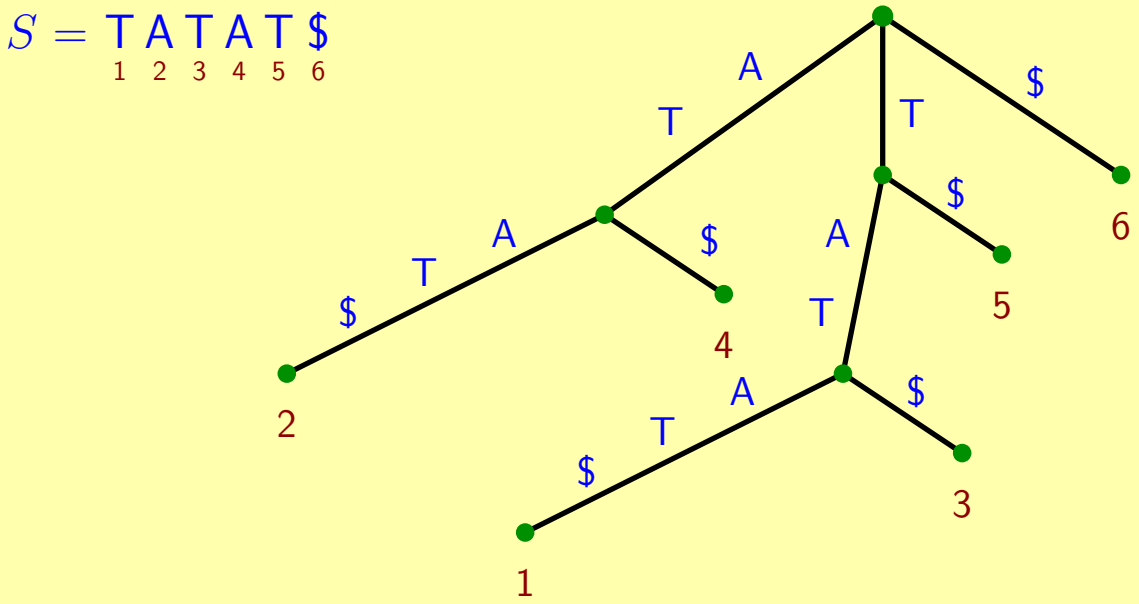


Suffix Tree: Definition

- A suffix of a string S of length n
  is a substring of S that ends at position n.

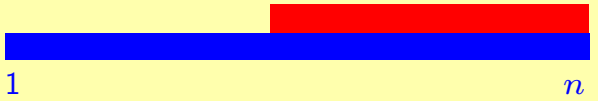


- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.



Suffix Tree: Definition

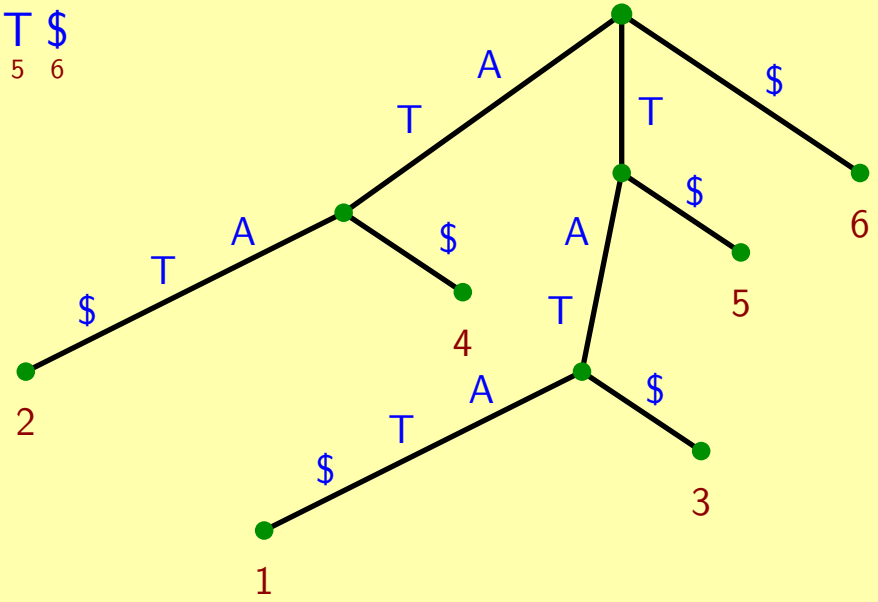
- A suffix of a string S of length n
  is a substring of S that ends at position n.



- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.

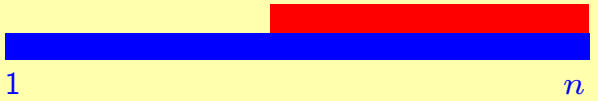
$S =$ T A T A T $
1 2 3 4 5 6

$P =$ T A T T



Suffix Tree: Definition

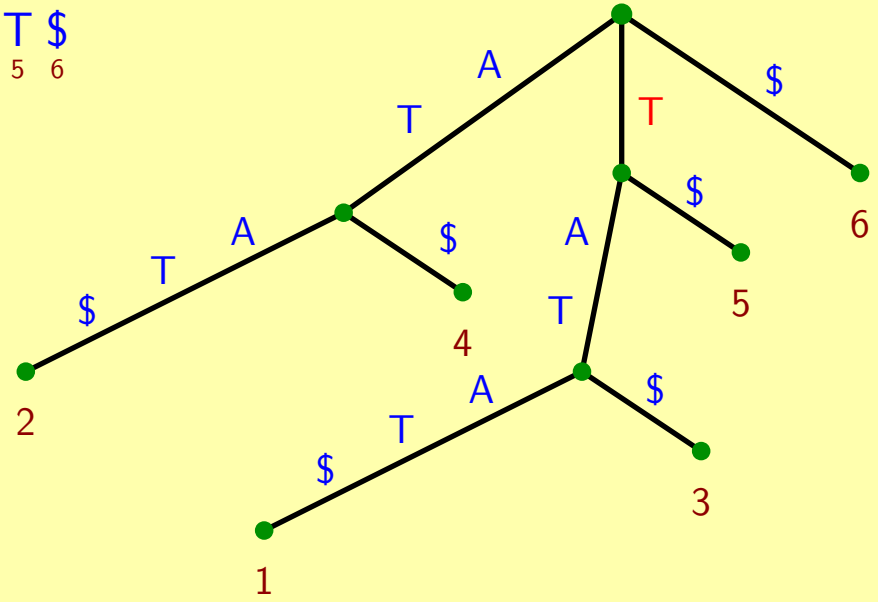
- A suffix of a string S of length n
  is a substring of S that ends at position n.



- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.

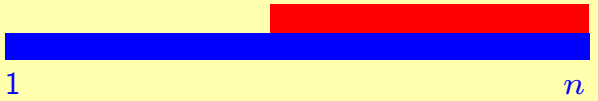
$S =$ T A T A T $
1 2 3 4 5 6

$P =$ T A T T



Suffix Tree: Definition

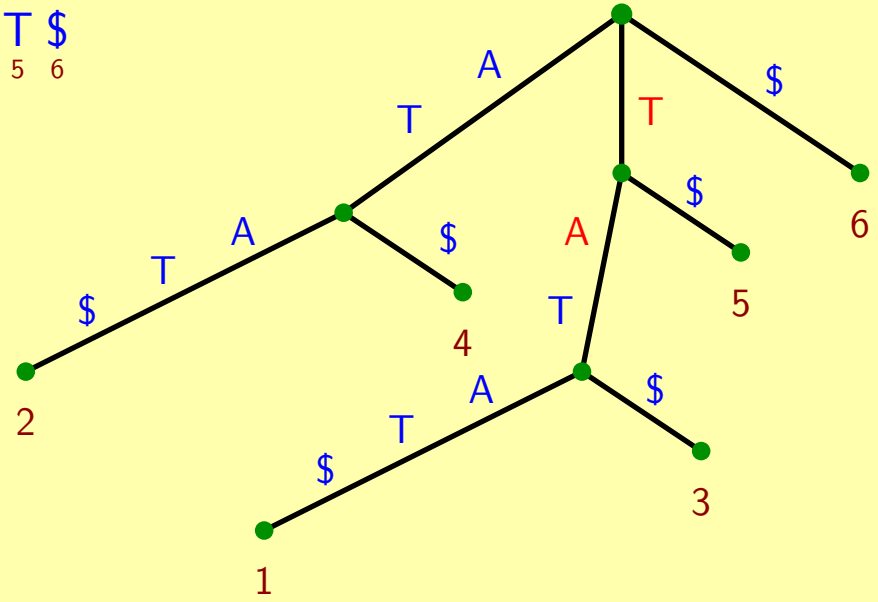
- A suffix of a string S of length n
  is a substring of S that ends at position n.



- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.

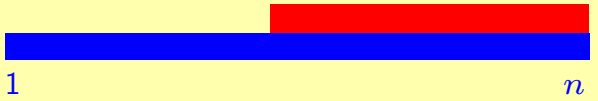
$S =$ T A T A T $
1 2 3 4 5 6

$P =$ T A T T



Suffix Tree: Definition

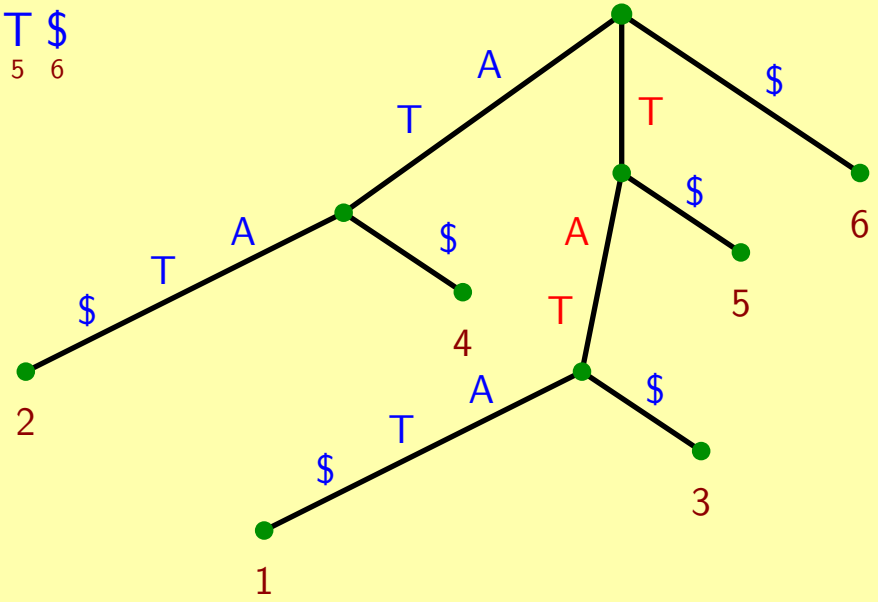
- A suffix of a string S of length n
  is a substring of S that ends at position n.



- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.

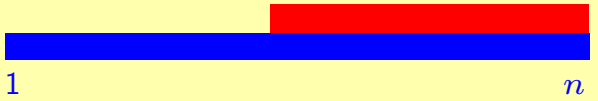
$S =$ T A T A T $
1 2 3 4 5 6

$P =$ T A T T



Suffix Tree: Definition

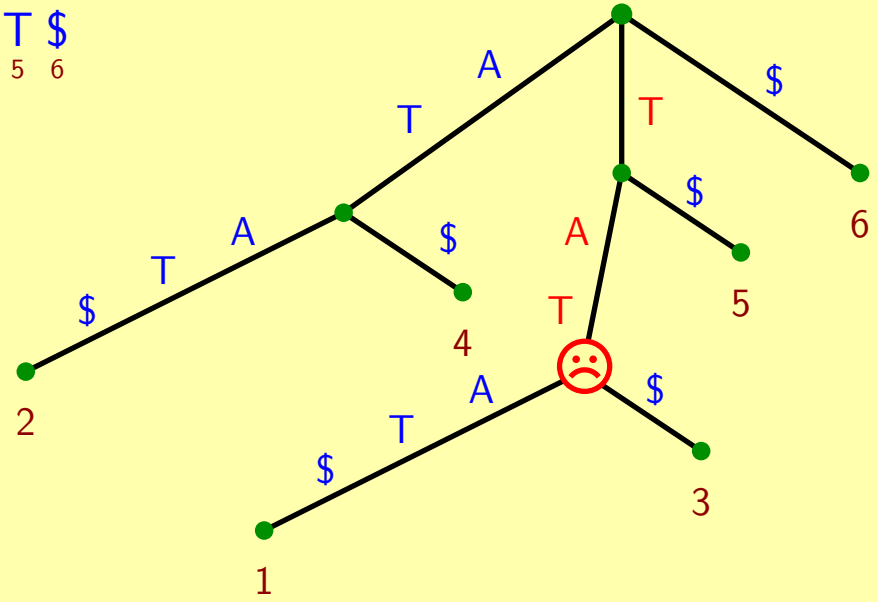
- A suffix of a string S of length n
  is a substring of S that ends at position n.



- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.

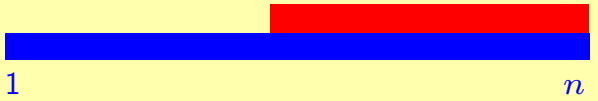
$S =$ T A T A T $
1 2 3 4 5 6

$P =$ T A T T

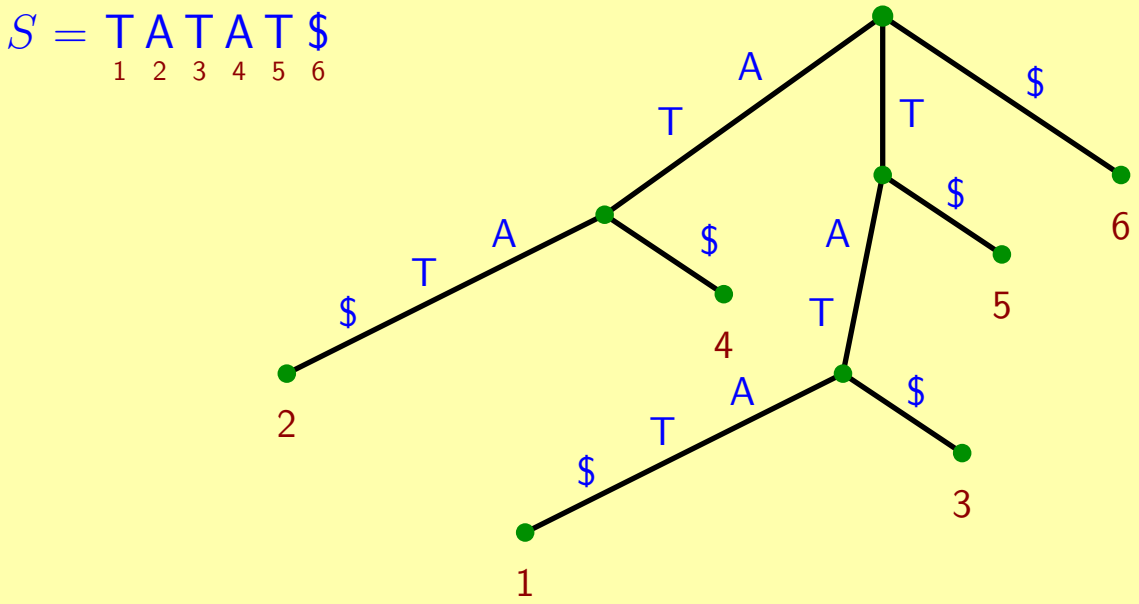


Suffix Tree: Definition

- A suffix of a string S of length n
  is a substring of S that ends at position n.



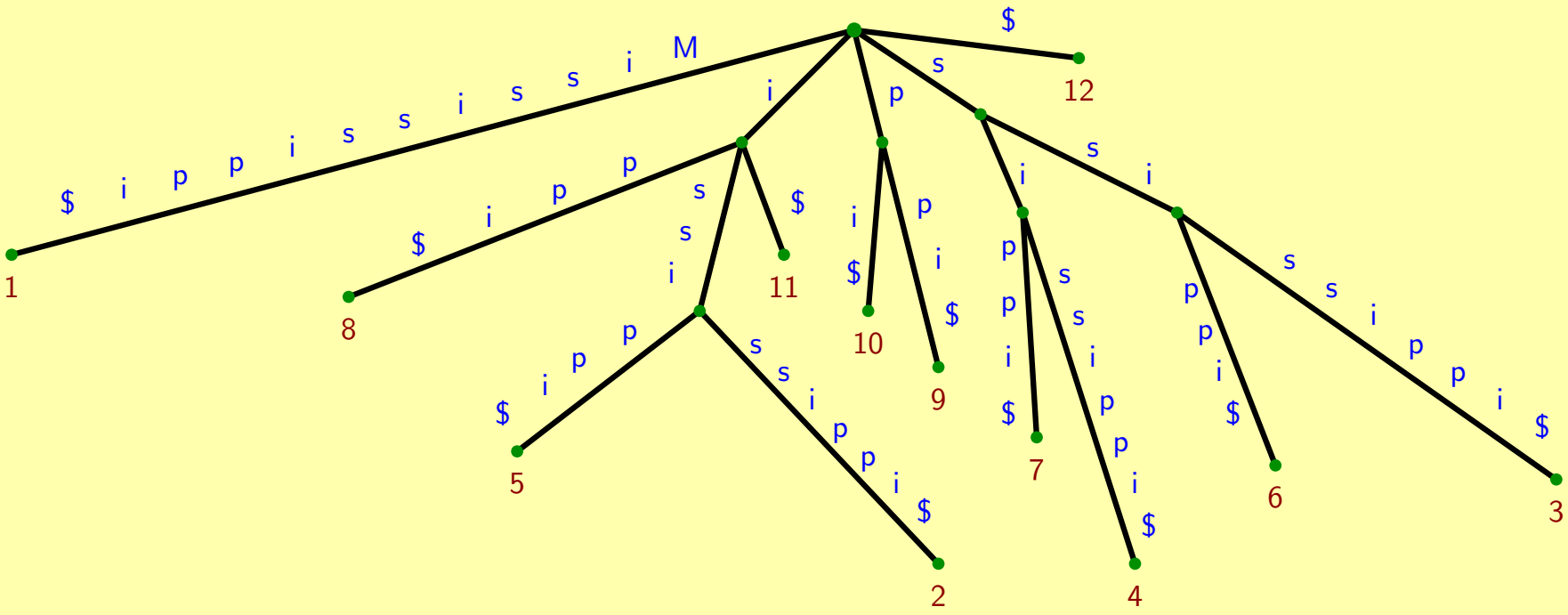
- The suffix tree of S, $T(S)$, is a rooted tree whose edges are labeled with strings such that
  - all edges leaving a node begin with different characters and
  - the paths from the root to the leaves represent all the suffixes of S.



A larger example

$S =$ M i s s i s s i p p i $

1 2 3 4 5 6 7 8 9 10 11 12



Suffix tree properties

- $T(S)$ represents exactly the substrings of S.
- $T(S)$ allows to enumerate these substrings and their locations in S in a convenient way.
- This is very useful for many pattern recognition problems, for example:
  - exact string matching as part of other applications, e.g. detecting DNA contamination
  - all-pairs suffix-prefix matching, important in fragment assembly
  - finding repeats and palindromes, tandem repeats, degenerate repeats
  - DNA primer design
  - DNA chip design
  - ...

See also:

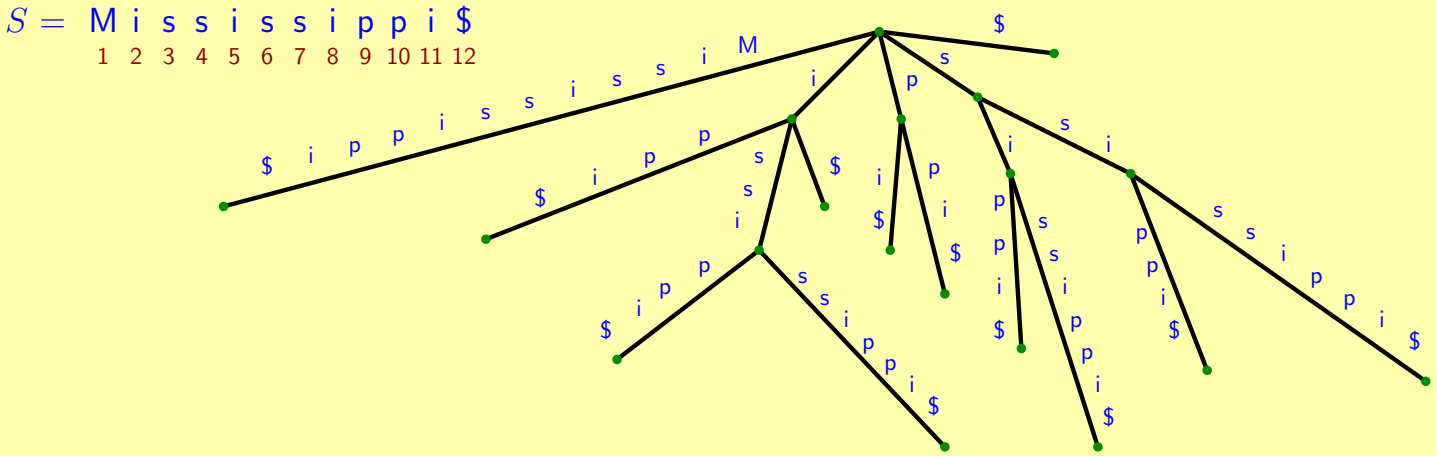
- A. Apostolico: The myriad virtues of subword trees, 1985.
- D. Gusfield: Algorithms on strings, trees, and sequences, 1997.

Space usage of suffix trees

Observation: $T(S)$ requires $\mathcal{O}(n)$ space.

Proof sketch:

1. $T(S)$ has at most n leaves.
2. Each internal node is branching $\Rightarrow$ at most $n-1$ internal nodes.
3. A tree with at most $2n-1$ nodes has at most $2n-2$ edges.
4. Each node requires constant space.
5. Each edge label is a substring of $S \Rightarrow$ pair of pointers (i, j) into S.

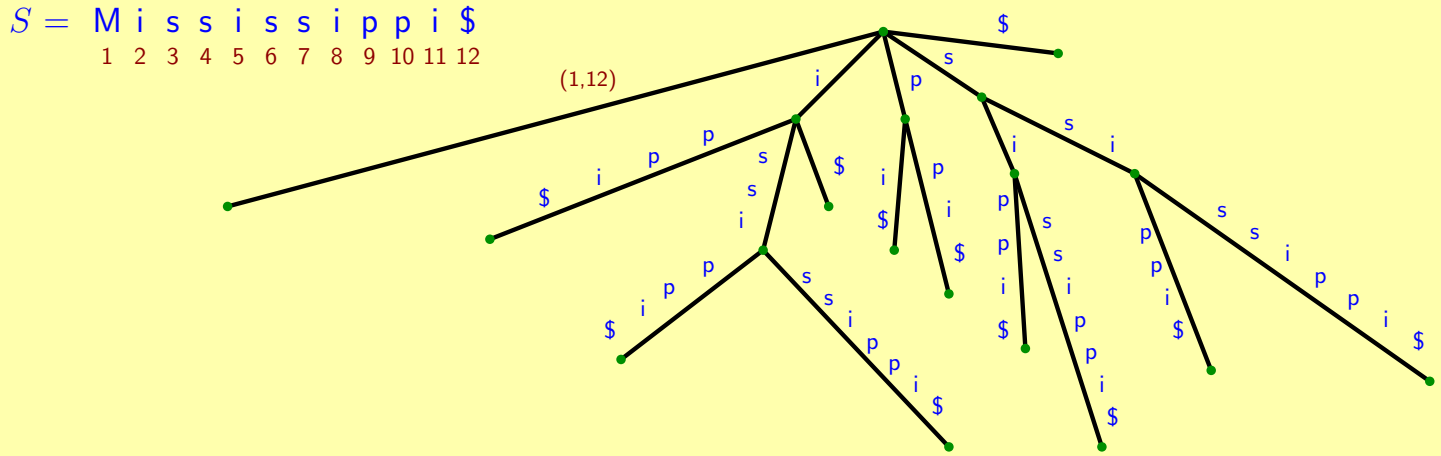


Space usage of suffix trees

Observation: $T(S)$ requires $\mathcal{O}(n)$ space.

Proof sketch:

1. $T(S)$ has at most n leaves.
2. Each internal node is branching $\Rightarrow$ at most $n-1$ internal nodes.
3. A tree with at most $2n-1$ nodes has at most $2n-2$ edges.
4. Each node requires constant space.
5. Each edge label is a substring of $S \Rightarrow$ pair of pointers (i, j) into S.

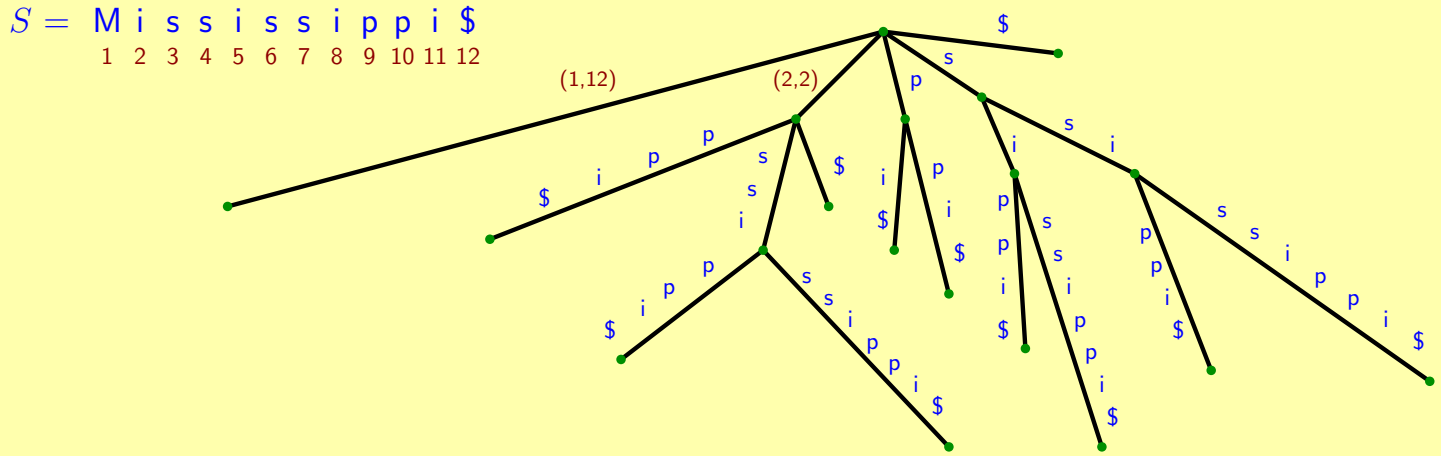


Space usage of suffix trees

Observation: $T(S)$ requires $\mathcal{O}(n)$ space.

Proof sketch:

1. $T(S)$ has at most n leaves.
2. Each internal node is branching $\Rightarrow$ at most $n-1$ internal nodes.
3. A tree with at most $2n-1$ nodes has at most $2n-2$ edges.
4. Each node requires constant space.
5. Each edge label is a substring of $S \Rightarrow$ pair of pointers (i, j) into S.

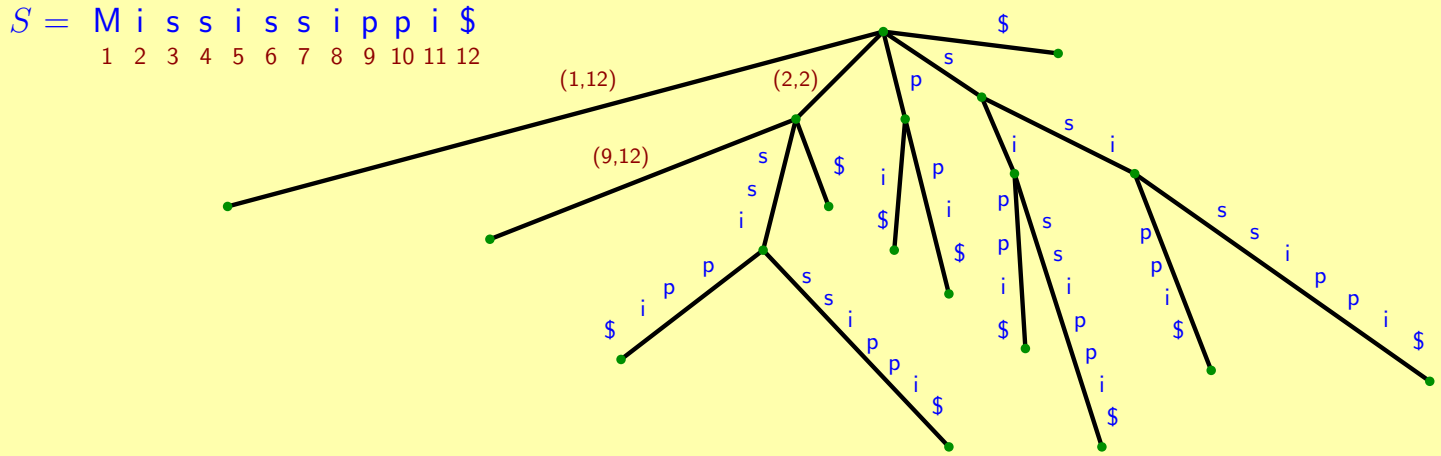


Space usage of suffix trees

Observation: $T(S)$ requires $\mathcal{O}(n)$ space.

Proof sketch:

1. $T(S)$ has at most n leaves.
2. Each internal node is branching $\Rightarrow$ at most $n-1$ internal nodes.
3. A tree with at most $2n-1$ nodes has at most $2n-2$ edges.
4. Each node requires constant space.
5. Each edge label is a substring of $S \Rightarrow$ pair of pointers (i, j) into S.

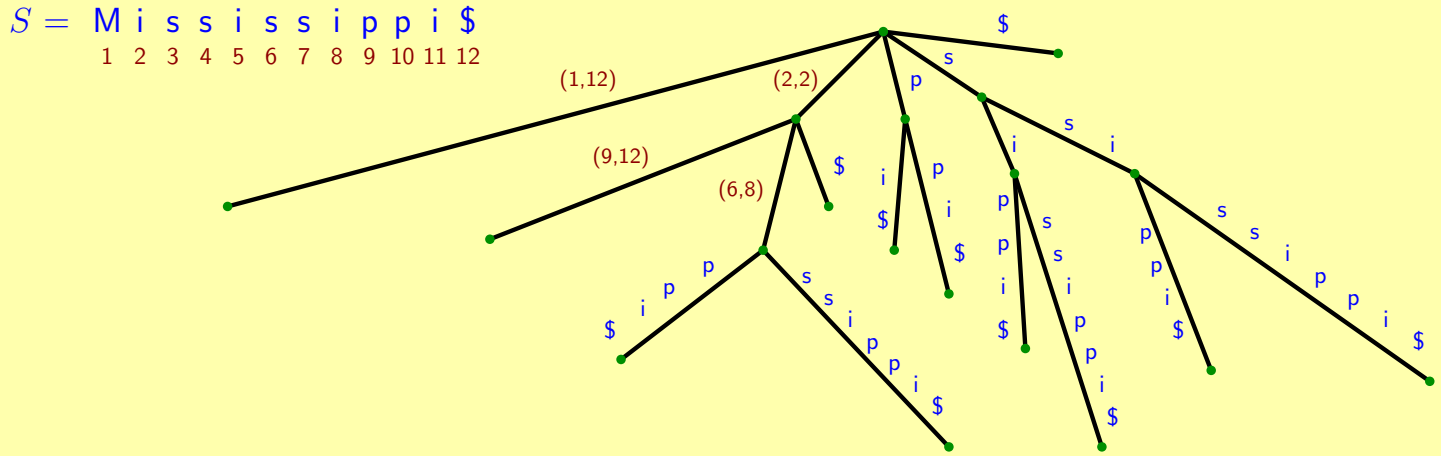


Space usage of suffix trees

Observation: $T(S)$ requires $\mathcal{O}(n)$ space.

Proof sketch:

1. $T(S)$ has at most n leaves.
2. Each internal node is branching $\Rightarrow$ at most $n-1$ internal nodes.
3. A tree with at most $2n-1$ nodes has at most $2n-2$ edges.
4. Each node requires constant space.
5. Each edge label is a substring of $S \Rightarrow$ pair of pointers (i, j) into S.

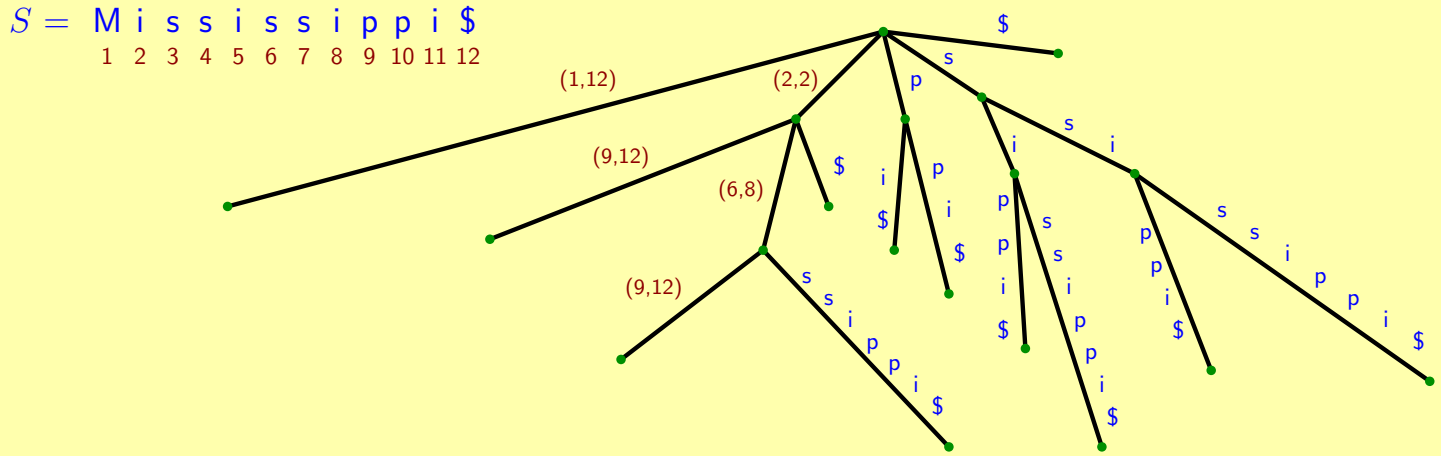


Space usage of suffix trees

Observation: $T(S)$ requires $\mathcal{O}(n)$ space.

Proof sketch:

1. $T(S)$ has at most n leaves.
2. Each internal node is branching $\Rightarrow$ at most $n-1$ internal nodes.
3. A tree with at most $2n-1$ nodes has at most $2n-2$ edges.
4. Each node requires constant space.
5. Each edge label is a substring of $S \Rightarrow$ pair of pointers (i, j) into S.

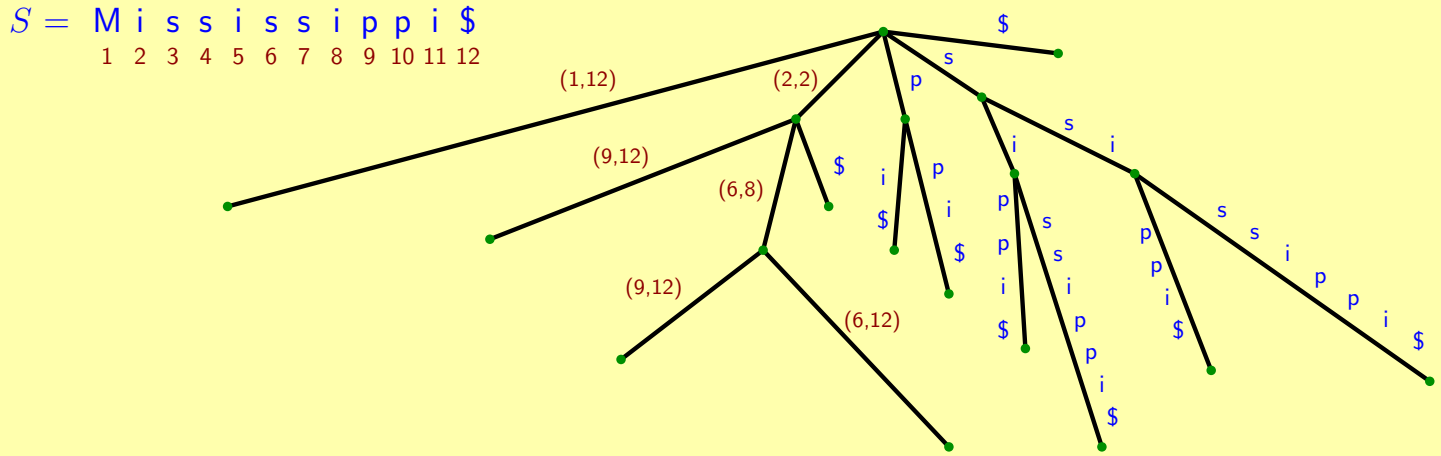


Space usage of suffix trees

Observation: $T(S)$ requires $\mathcal{O}(n)$ space.

Proof sketch:

1. $T(S)$ has at most n leaves.
2. Each internal node is branching $\Rightarrow$ at most $n-1$ internal nodes.
3. A tree with at most $2n-1$ nodes has at most $2n-2$ edges.
4. Each node requires constant space.
5. Each edge label is a substring of $S \Rightarrow$ pair of pointers (i, j) into S.

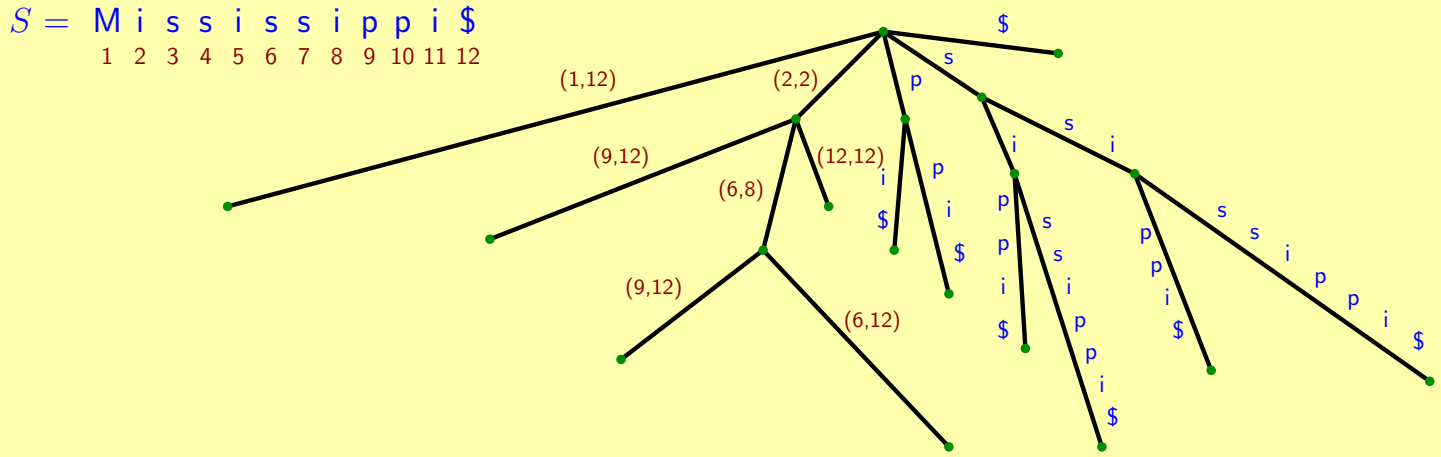


Space usage of suffix trees

Observation: $T(S)$ requires $\mathcal{O}(n)$ space.

Proof sketch:

1. $T(S)$ has at most n leaves.
2. Each internal node is branching $\Rightarrow$ at most $n-1$ internal nodes.
3. A tree with at most $2n-1$ nodes has at most $2n-2$ edges.
4. Each node requires constant space.
5. Each edge label is a substring of $S \Rightarrow$ pair of pointers (i, j) into S.

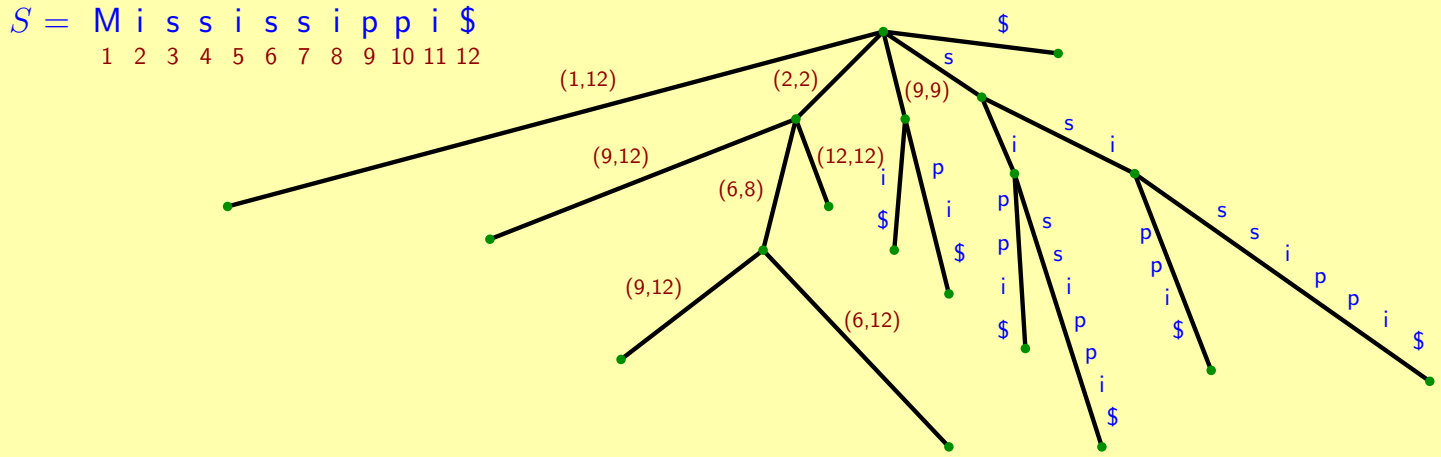


Space usage of suffix trees

Observation: $T(S)$ requires $\mathcal{O}(n)$ space.

Proof sketch:

1. $T(S)$ has at most n leaves.
2. Each internal node is branching $\Rightarrow$ at most $n-1$ internal nodes.
3. A tree with at most $2n-1$ nodes has at most $2n-2$ edges.
4. Each node requires constant space.
5. Each edge label is a substring of $S \Rightarrow$ pair of pointers (i, j) into S.

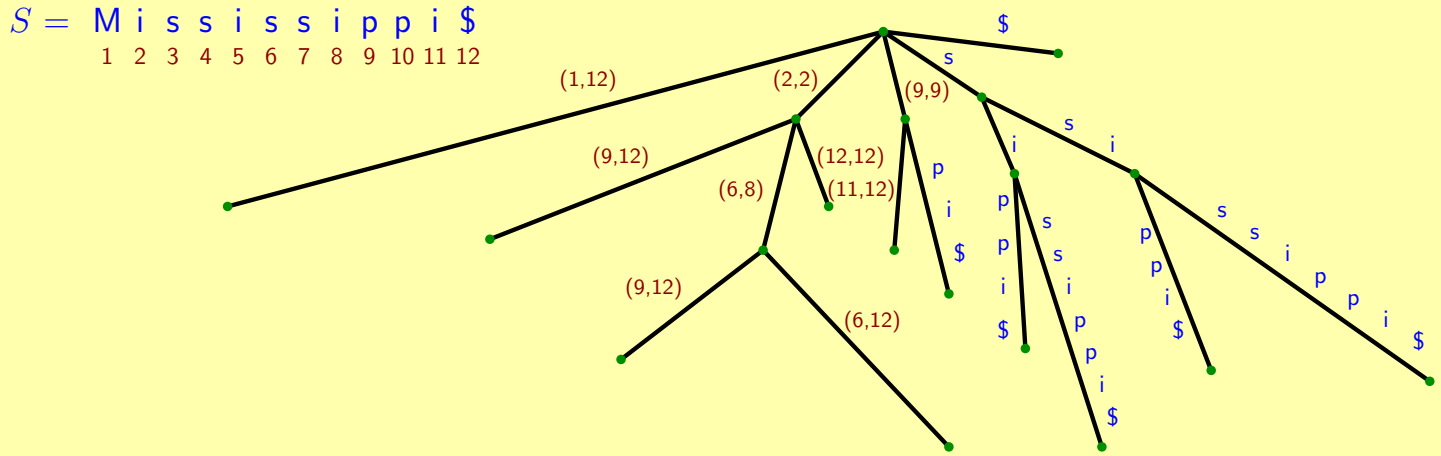


Space usage of suffix trees

Observation: $T(S)$ requires $\mathcal{O}(n)$ space.

Proof sketch:

1. $T(S)$ has at most n leaves.
2. Each internal node is branching $\Rightarrow$ at most $n-1$ internal nodes.
3. A tree with at most $2n-1$ nodes has at most $2n-2$ edges.
4. Each node requires constant space.
5. Each edge label is a substring of $S \Rightarrow$ pair of pointers (i, j) into S.

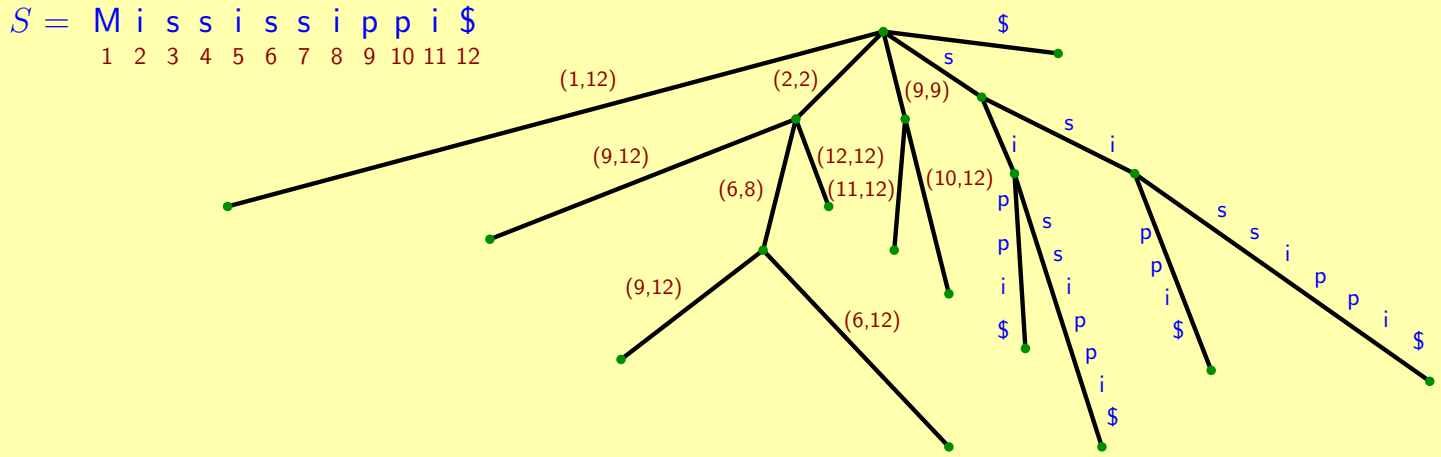


Space usage of suffix trees

Observation: $T(S)$ requires $\mathcal{O}(n)$ space.

Proof sketch:

1. $T(S)$ has at most n leaves.
2. Each internal node is branching $\Rightarrow$ at most $n-1$ internal nodes.
3. A tree with at most $2n-1$ nodes has at most $2n-2$ edges.
4. Each node requires constant space.
5. Each edge label is a substring of $S \Rightarrow$ pair of pointers (i, j) into S.

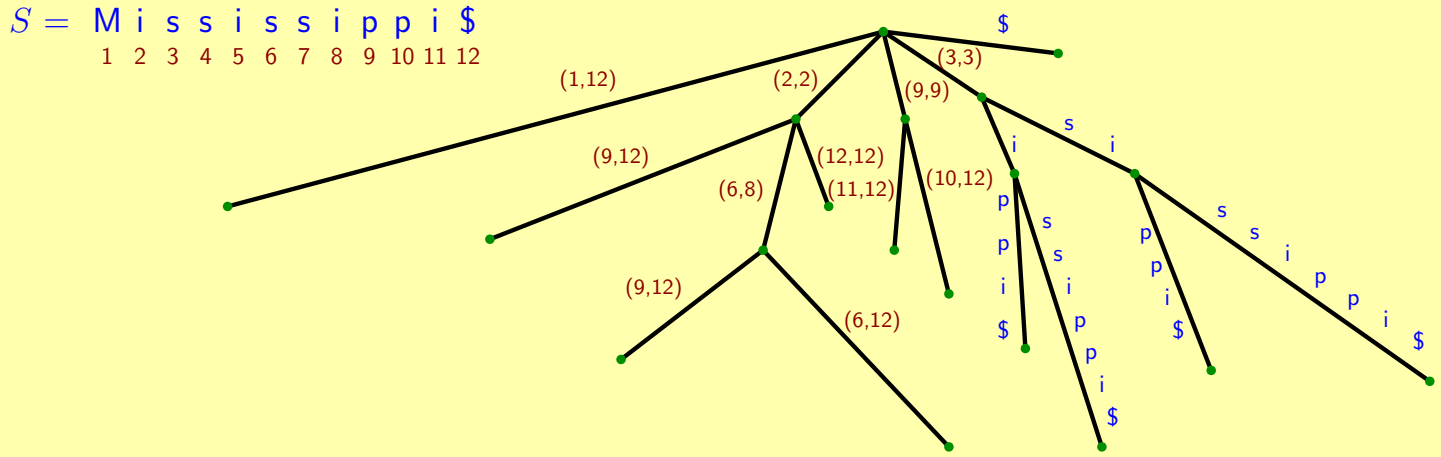


Space usage of suffix trees

Observation: $T(S)$ requires $\mathcal{O}(n)$ space.

Proof sketch:

1. $T(S)$ has at most n leaves.
2. Each internal node is branching $\Rightarrow$ at most $n-1$ internal nodes.
3. A tree with at most $2n-1$ nodes has at most $2n-2$ edges.
4. Each node requires constant space.
5. Each edge label is a substring of $S \Rightarrow$ pair of pointers (i, j) into S.

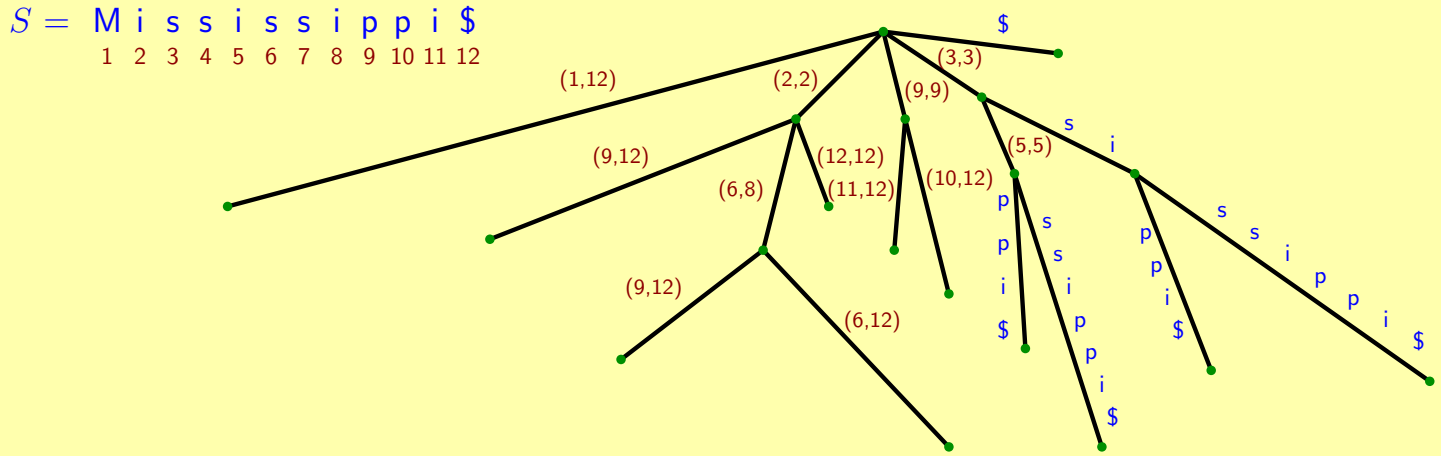


Space usage of suffix trees

Observation: $T(S)$ requires $\mathcal{O}(n)$ space.

Proof sketch:

1. $T(S)$ has at most n leaves.
2. Each internal node is branching $\Rightarrow$ at most $n-1$ internal nodes.
3. A tree with at most $2n-1$ nodes has at most $2n-2$ edges.
4. Each node requires constant space.
5. Each edge label is a substring of $S \Rightarrow$ pair of pointers (i, j) into S.

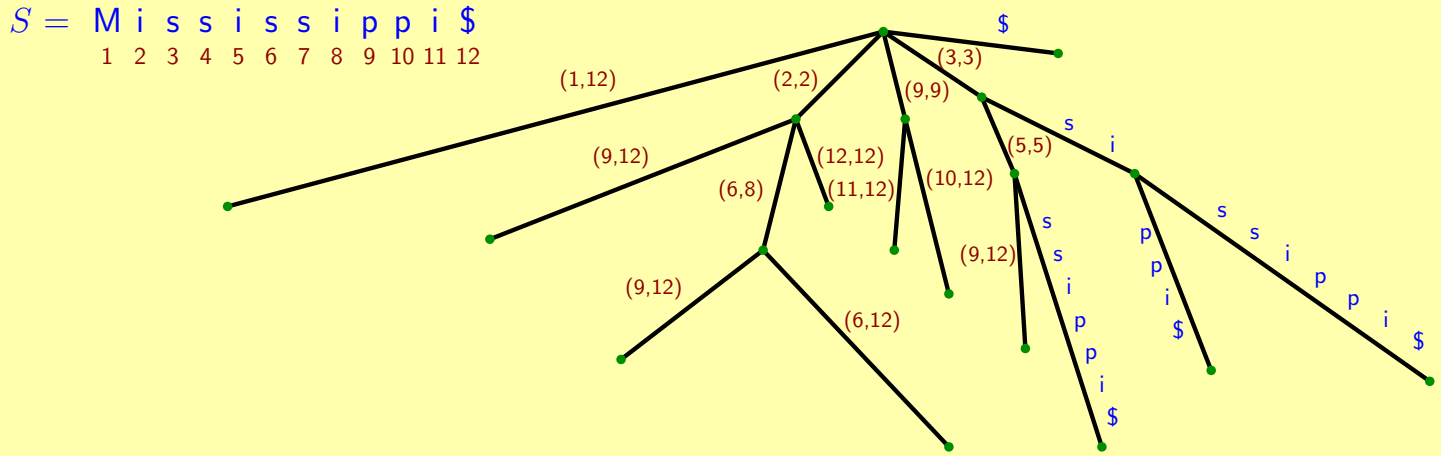


Space usage of suffix trees

Observation: $T(S)$ requires $\mathcal{O}(n)$ space.

Proof sketch:

1. $T(S)$ has at most n leaves.
2. Each internal node is branching $\Rightarrow$ at most $n-1$ internal nodes.
3. A tree with at most $2n-1$ nodes has at most $2n-2$ edges.
4. Each node requires constant space.
5. Each edge label is a substring of $S \Rightarrow$ pair of pointers (i, j) into S.

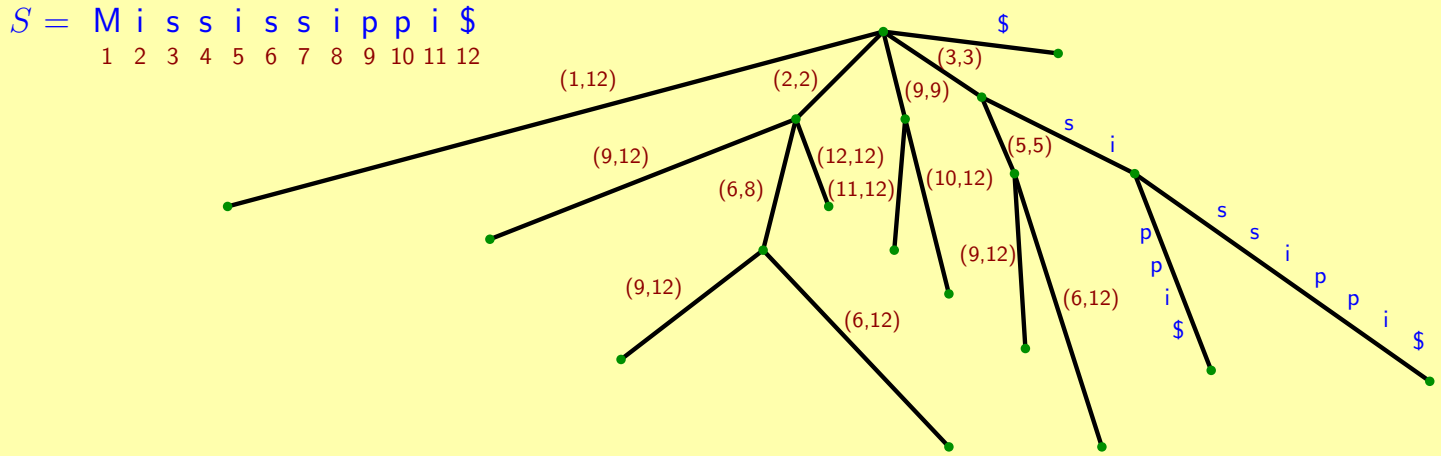


Space usage of suffix trees

Observation: $T(S)$ requires $\mathcal{O}(n)$ space.

Proof sketch:

1. $T(S)$ has at most n leaves.
2. Each internal node is branching $\Rightarrow$ at most $n-1$ internal nodes.
3. A tree with at most $2n-1$ nodes has at most $2n-2$ edges.
4. Each node requires constant space.
5. Each edge label is a substring of $S \Rightarrow$ pair of pointers (i, j) into S.

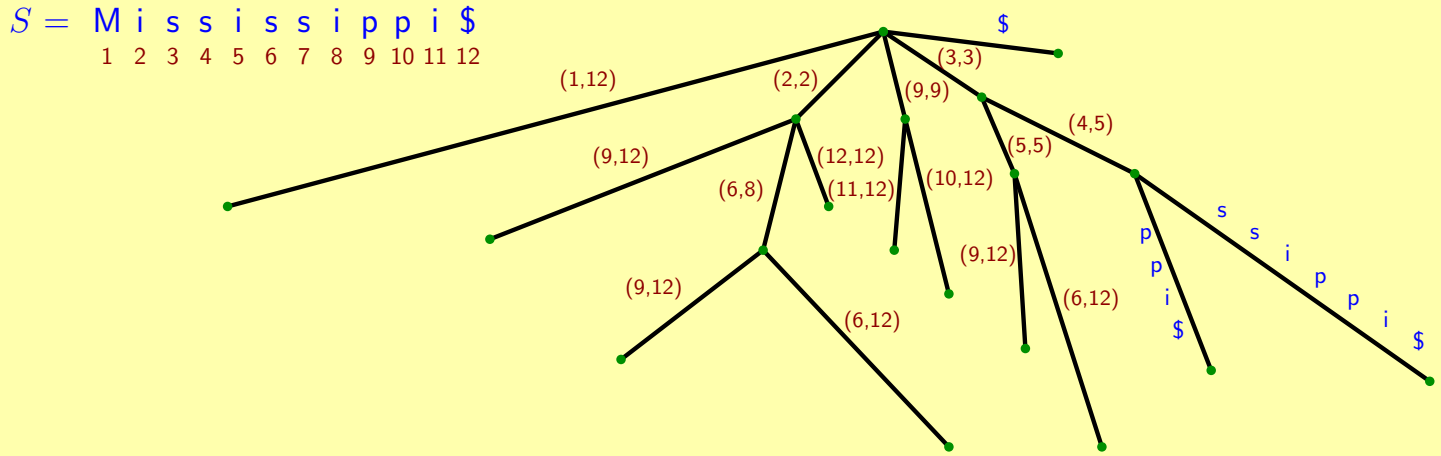


Space usage of suffix trees

Observation: $T(S)$ requires $\mathcal{O}(n)$ space.

Proof sketch:

1. $T(S)$ has at most n leaves.
2. Each internal node is branching $\Rightarrow$ at most $n-1$ internal nodes.
3. A tree with at most $2n-1$ nodes has at most $2n-2$ edges.
4. Each node requires constant space.
5. Each edge label is a substring of $S \Rightarrow$ pair of pointers (i, j) into S.

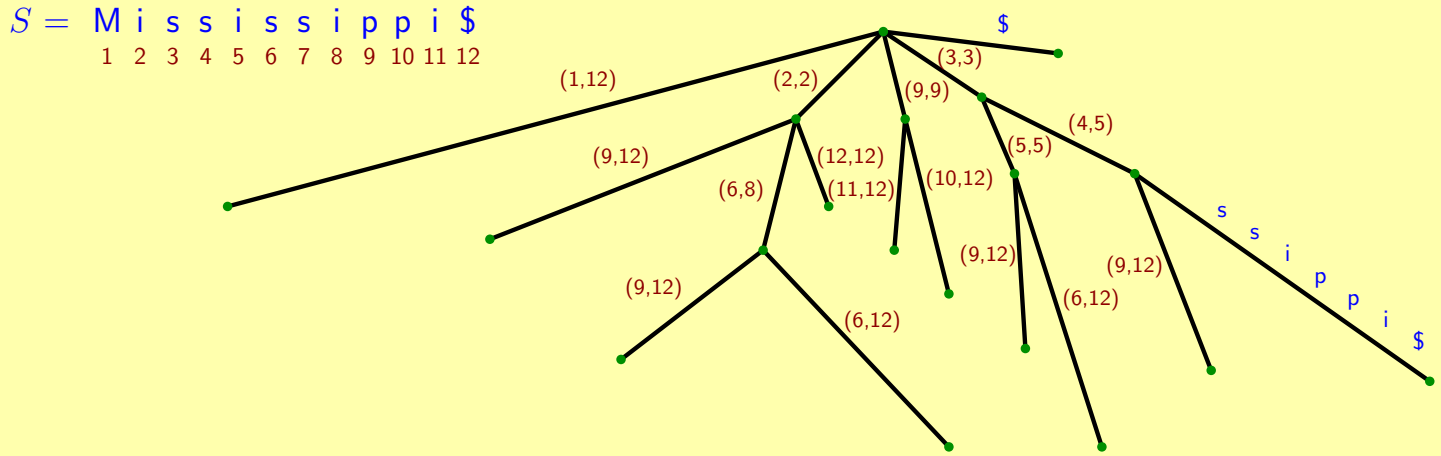


Space usage of suffix trees

Observation: $T(S)$ requires $\mathcal{O}(n)$ space.

Proof sketch:

1. $T(S)$ has at most n leaves.
2. Each internal node is branching $\Rightarrow$ at most $n-1$ internal nodes.
3. A tree with at most $2n-1$ nodes has at most $2n-2$ edges.
4. Each node requires constant space.
5. Each edge label is a substring of $S \Rightarrow$ pair of pointers (i, j) into S.

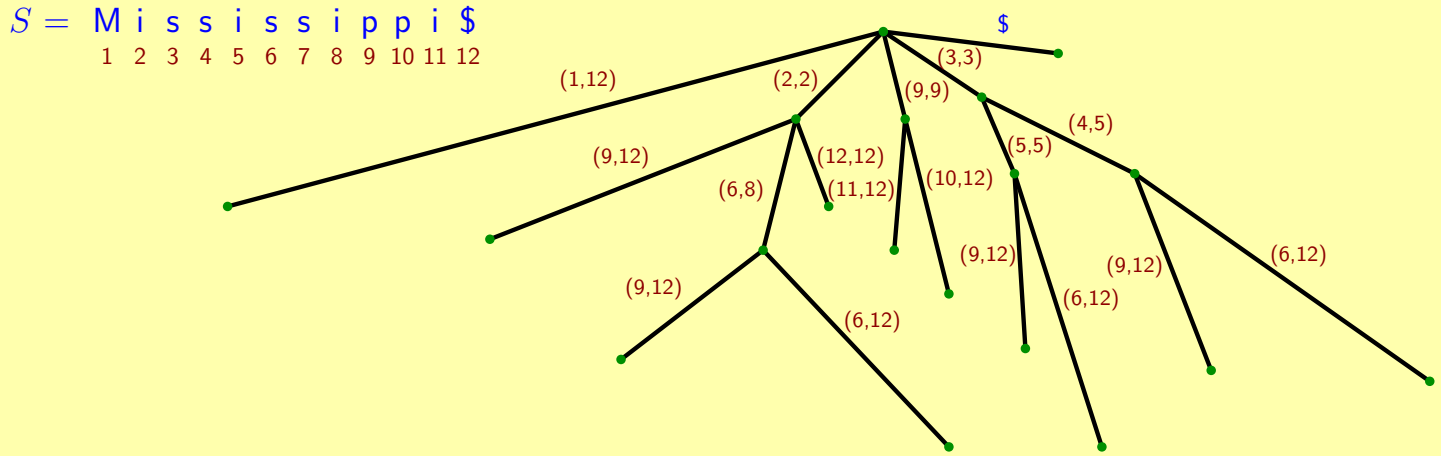


Space usage of suffix trees

Observation: $T(S)$ requires $\mathcal{O}(n)$ space.

Proof sketch:

1. $T(S)$ has at most n leaves.
2. Each internal node is branching $\Rightarrow$ at most $n-1$ internal nodes.
3. A tree with at most $2n-1$ nodes has at most $2n-2$ edges.
4. Each node requires constant space.
5. Each edge label is a substring of $S \Rightarrow$ pair of pointers (i, j) into S.

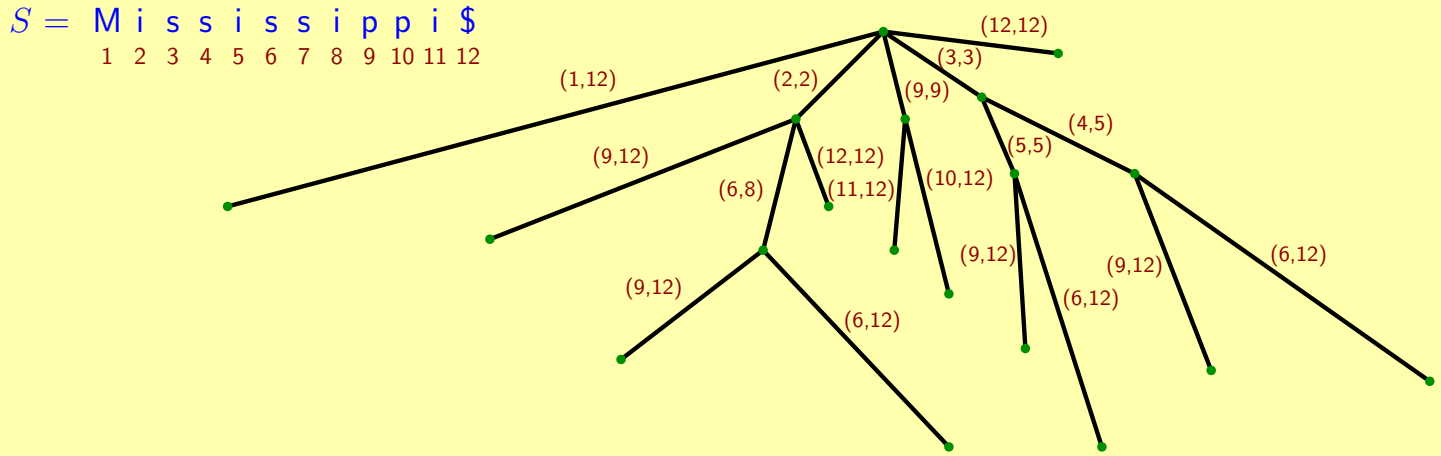


Space usage of suffix trees

Observation: $T(S)$ requires $\mathcal{O}(n)$ space.

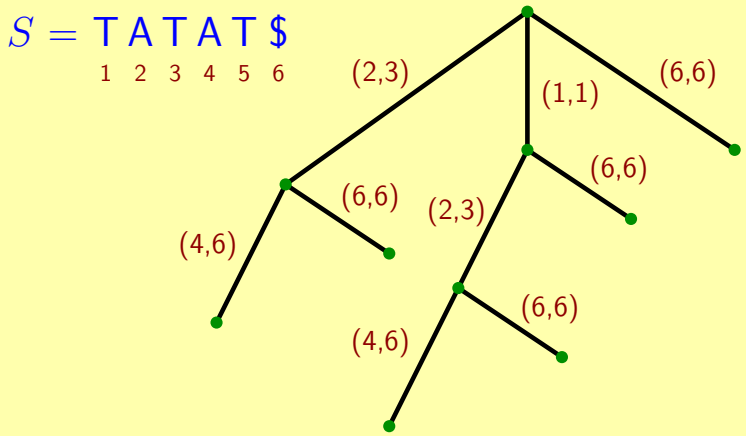
Proof sketch:

1. $T(S)$ has at most n leaves.
2. Each internal node is branching $\Rightarrow$ at most $n-1$ internal nodes.
3. A tree with at most $2n-1$ nodes has at most $2n-2$ edges.
4. Each node requires constant space.
5. Each edge label is a substring of $S \Rightarrow$ pair of pointers (i, j) into S.

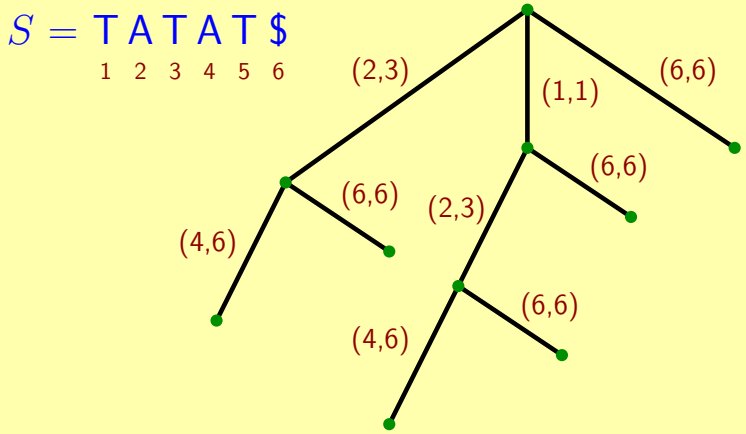


Representation of suffix trees

Representation of suffix trees



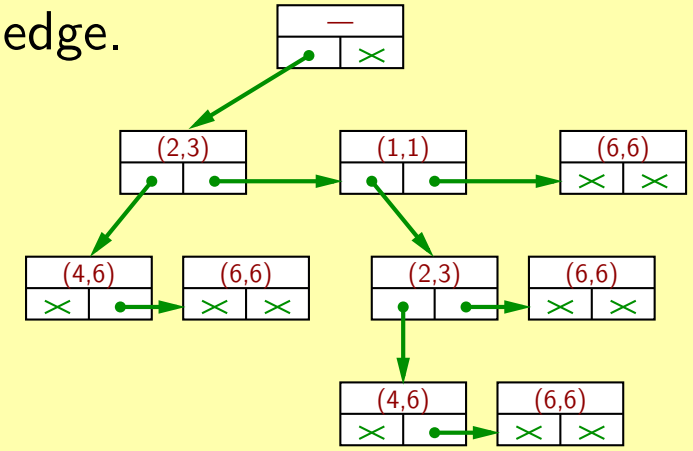
Representation of suffix trees



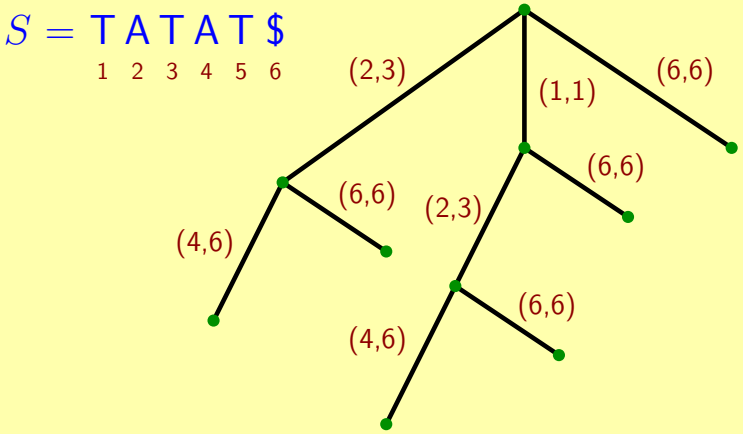
Standard representation of trees:

- Store nodes as records with child and sibling pointer.
- An edge label (i, j) is stored at node below the edge.

$\Rightarrow$ about $32n$ bytes in the worst case
$2n$ nodes $\times$ (2 integers $+$ 2 pointers)



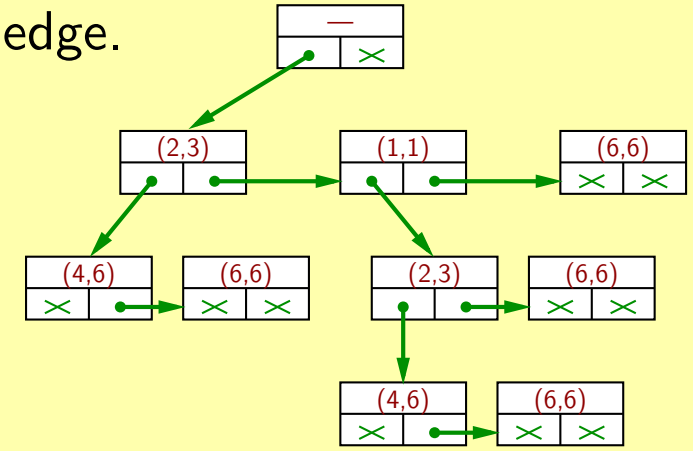
Representation of suffix trees



Standard representation of trees:

- Store nodes as records with child and sibling pointer.
- An edge label (i, j) is stored at node below the edge.

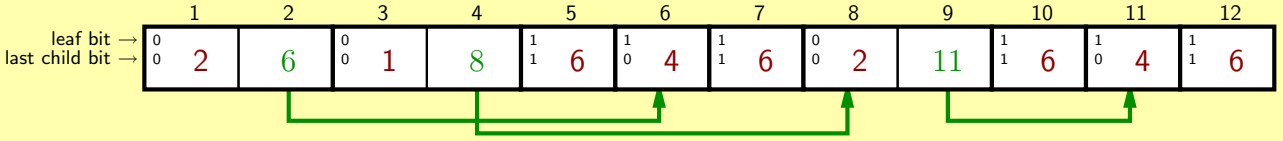
$\Rightarrow$ about $32n$ bytes in the worst case
$2n$ nodes $\times$ (2 integers + 2 pointers)



Ideas for more efficient representation:

- Do not represent leaves explicitly.
- Avoid sibling pointers by storing all children of the same node in a row.
- Do not represent the right pointer of an edge label.

$\Rightarrow$ below $12n$ bytes in the worst case, $8.5n$ on average



Construction of suffix trees

Construction of suffix trees

Theorem [Weiner, 1973]: $T(S)$ can be constructed in $\mathcal{O}(n)$ time.

There are two practical algorithms that construct the suffix tree in linear time: McCreight (1976) and Ukkonen (1993).

Construction of suffix trees

Theorem [Weiner, 1973]: $T(S)$ can be constructed in $\mathcal{O}(n)$ time.

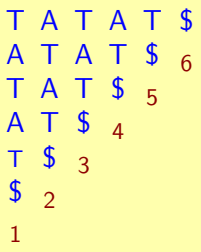
There are two practical algorithms that construct the suffix tree in linear time: McCreight (1976) and Ukkonen (1993).

A simpler algorithm is the WOTD (write-only, top-down) algorithm:

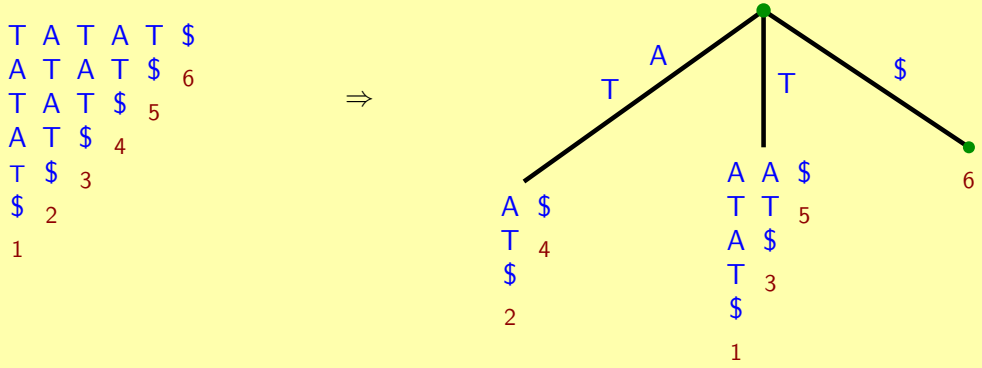
1. Let X be the set of all suffixes of S.
2. Sort the suffixes in X according to their first character.
3. For each group X_c ($c \in \Sigma$):
   (i) if X_c is a singleton, create a leaf;
   (ii) otherwise, find the longest common prefix of the suffixes in X_c, create an internal node, and recursively continue with Step 2, X being the set of remaining suffixes from X_c after splitting off the longest common prefix.

Analysis: $\mathcal{O}(n^2)$ worst-case time, $\mathcal{O}(n \log n)$ expected time, $\mathcal{O}(n)$ space

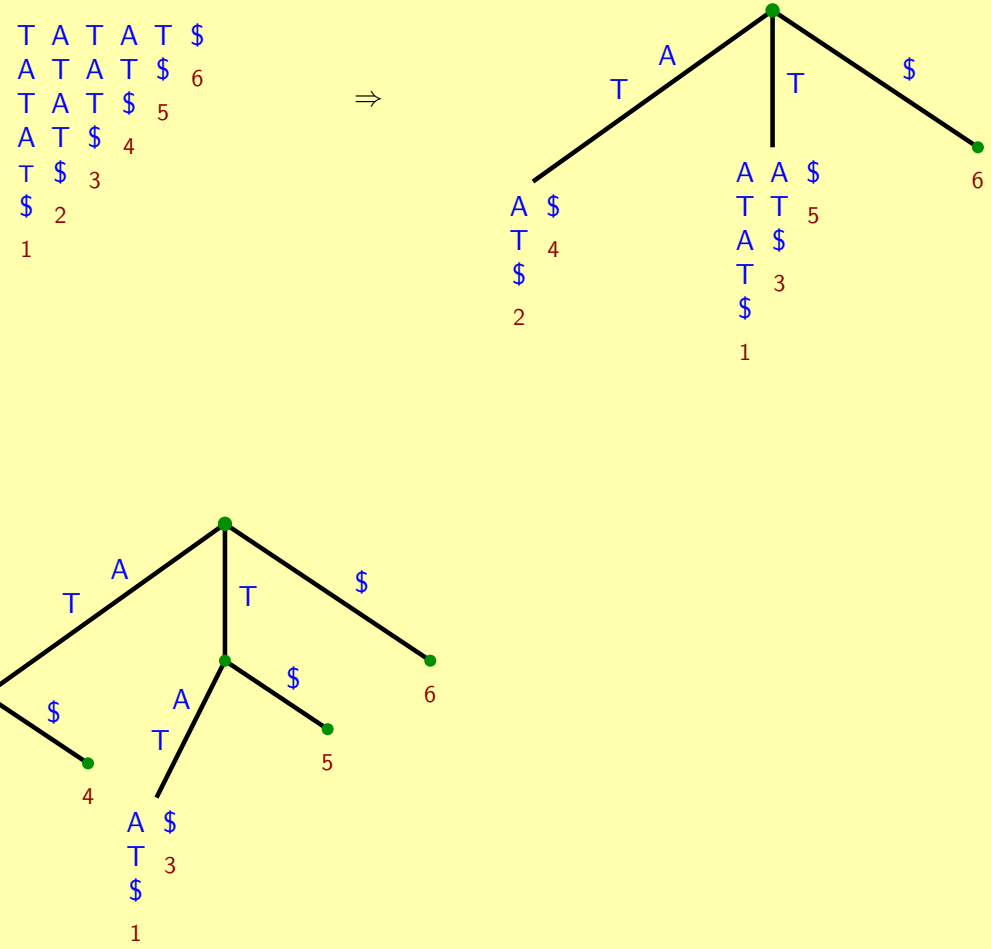
The WOTD construction algorithm



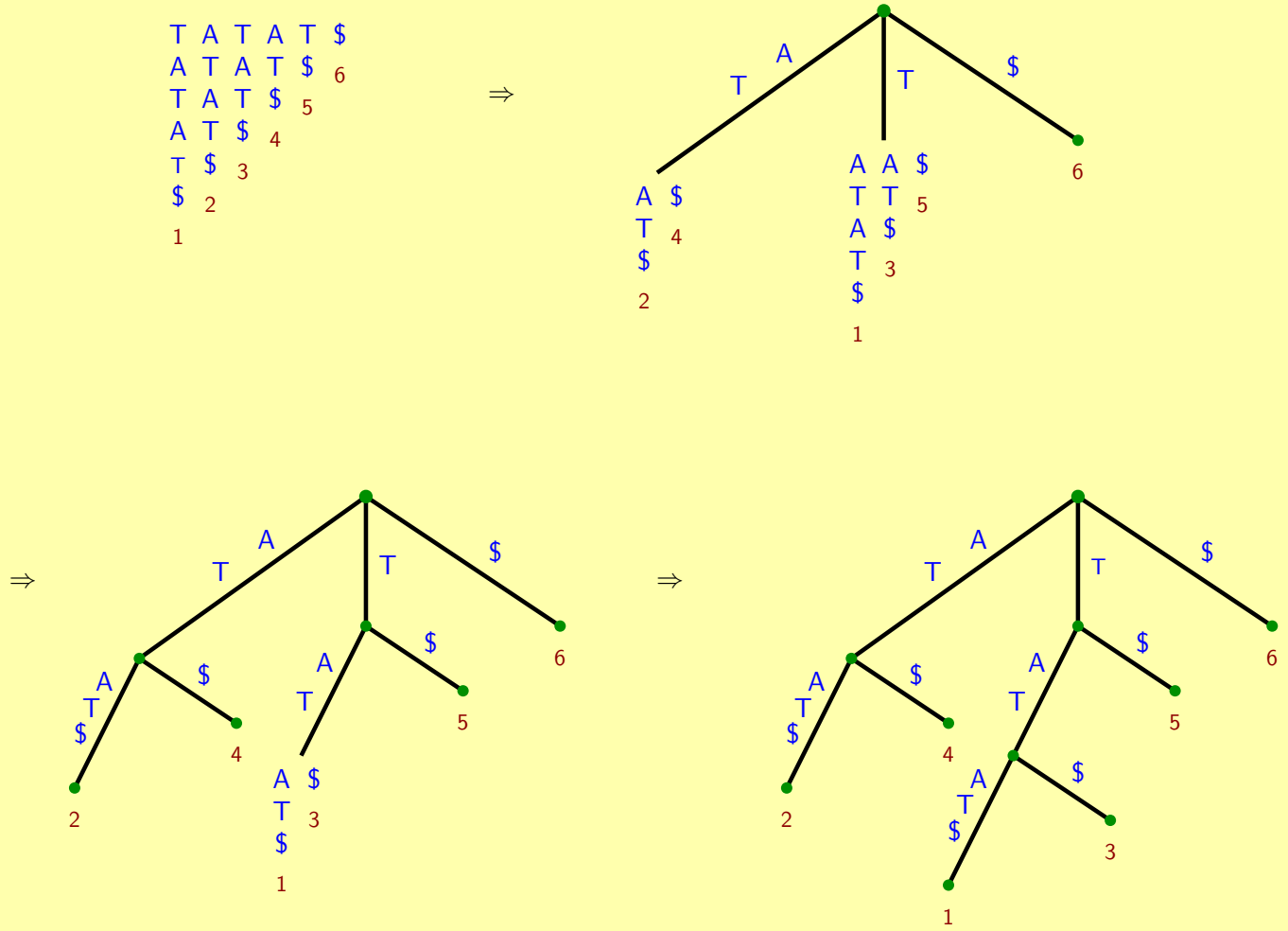
The WOTD construction algorithm



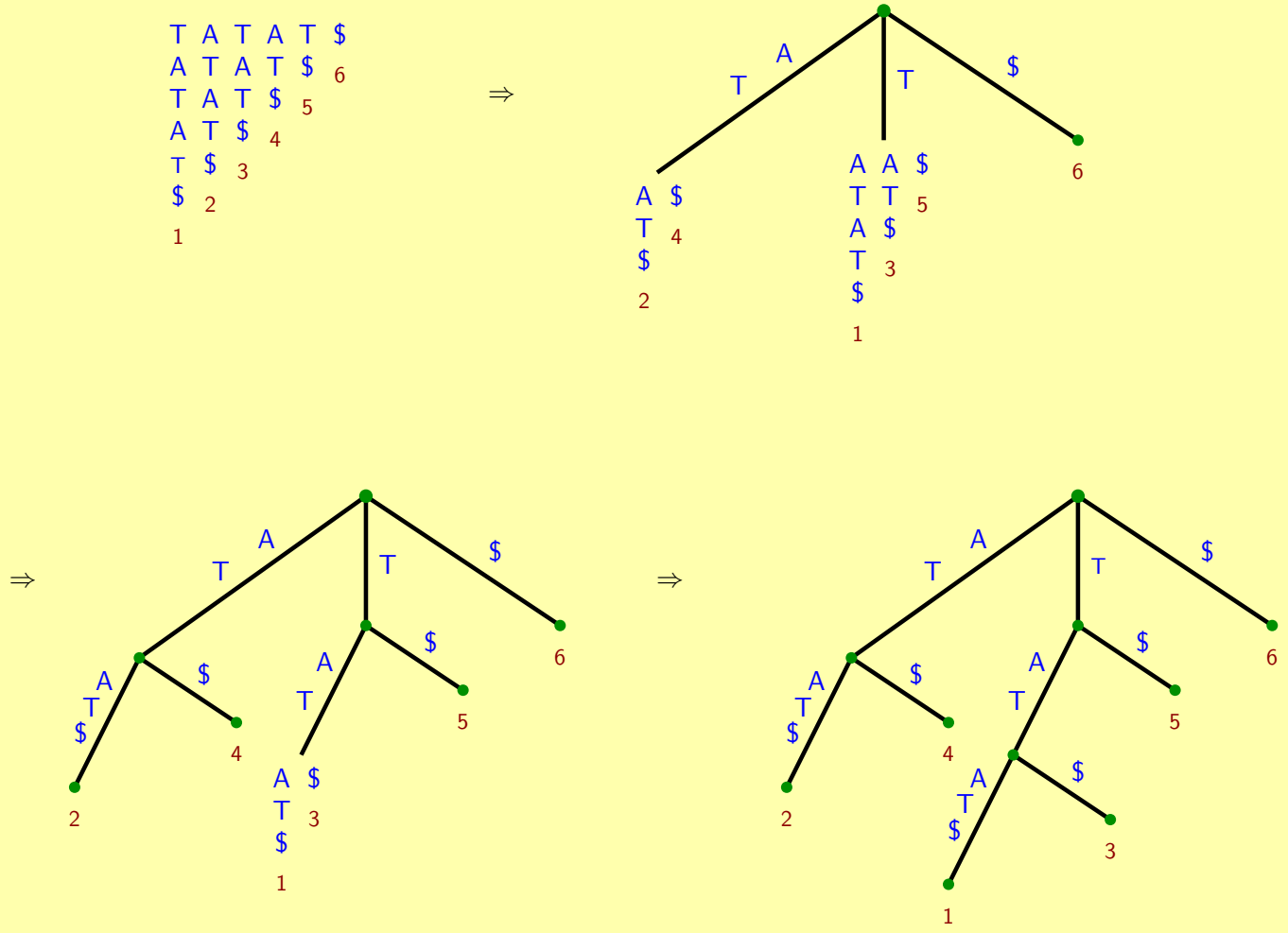
The WOTD construction algorithm



The WOTD construction algorithm



The WOTD construction algorithm



Note: The WOTD algorithm is well suited for a lazy construction of suffix trees.

Comparison: Exact string matching online and offline

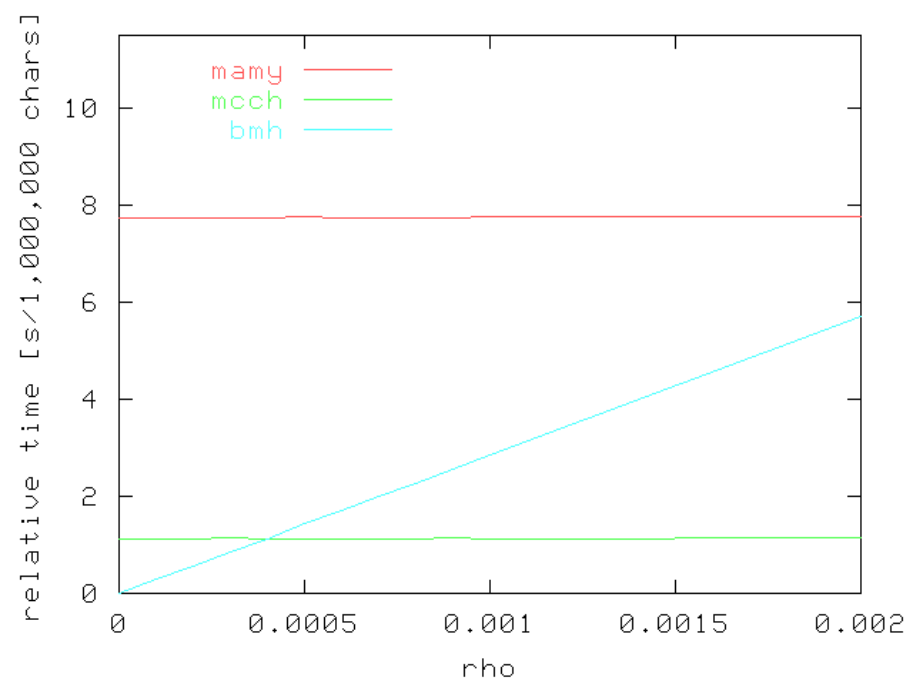
Theoretical results:

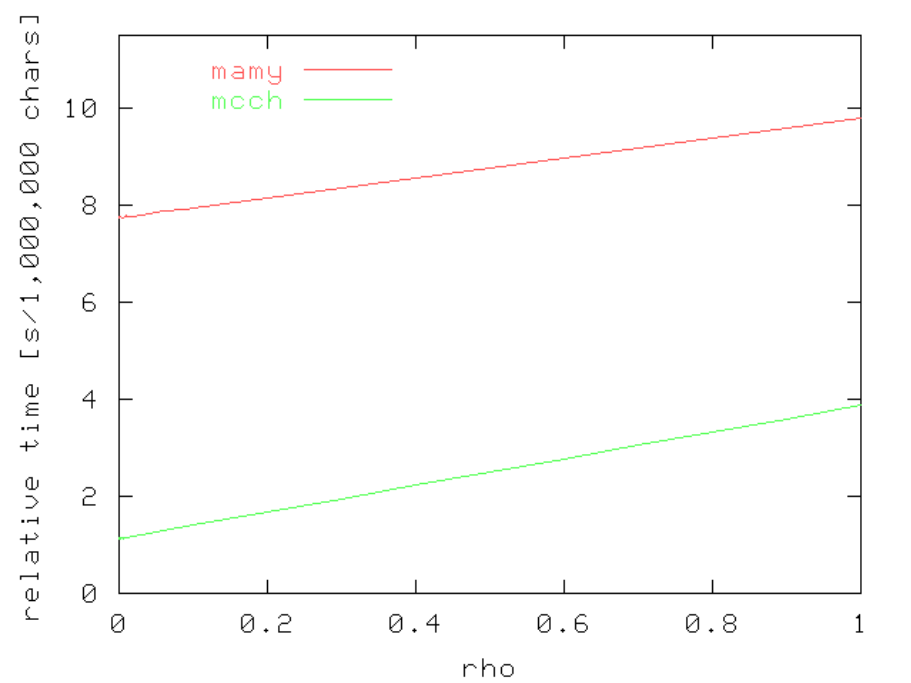
	online (no preprocessing)	offline (suffixtree)
1 pattern search	$\mathcal{O}(n+m)$	$\mathcal{O}(n+m)$
k pattern searches	$\mathcal{O}(k\ (n+m))$	$\mathcal{O}(n+km)$

n = text length $\qquad$ m = pattern length

Comparison: Exact string matching online and offline

Experimental results: index construction plus ρn pattern searches for $\rho \in [0, 1]$

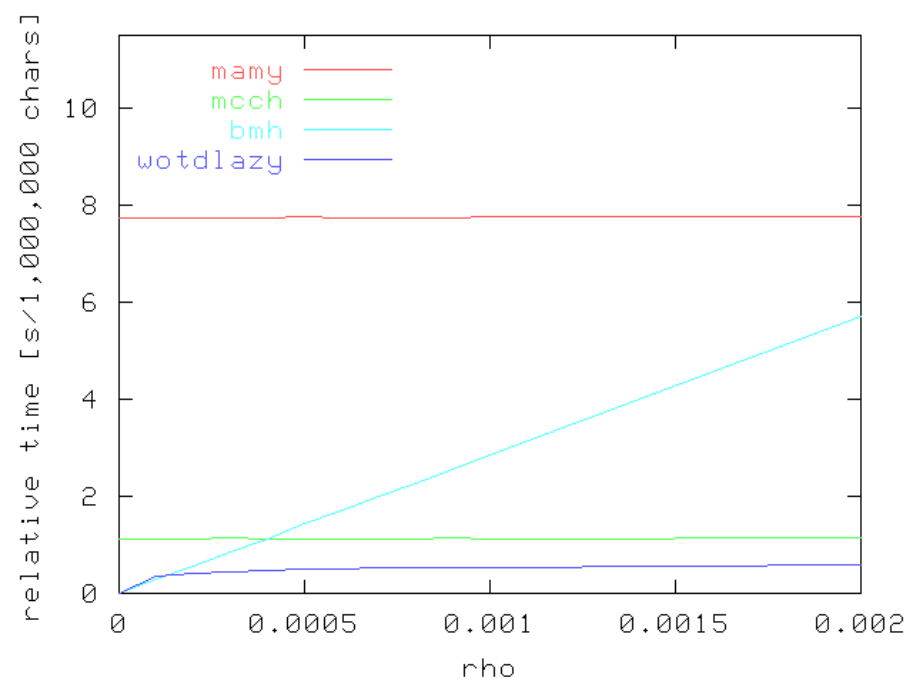


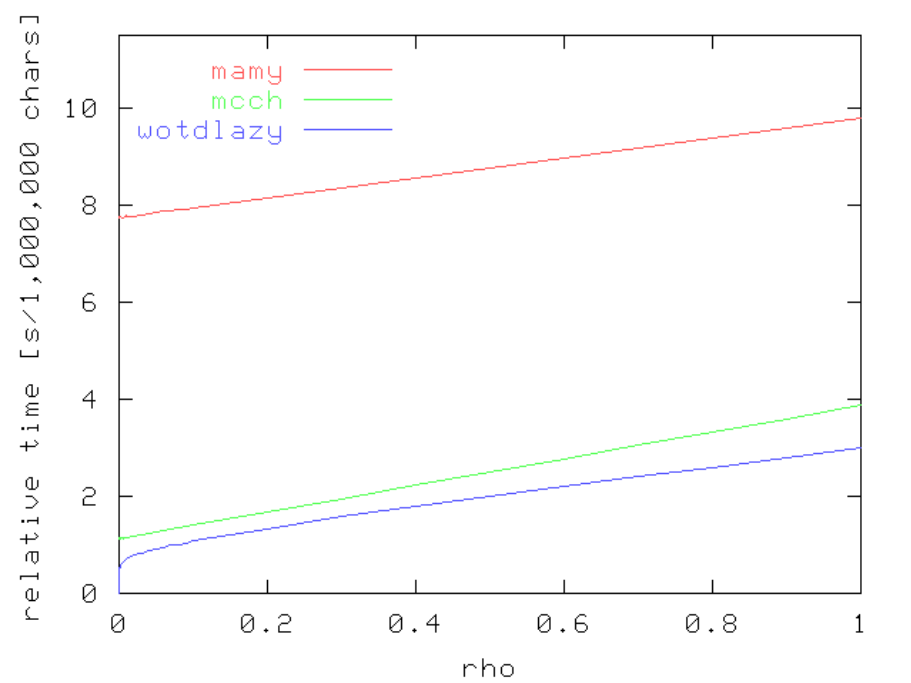


mamy = suffix array (implementation by Manber/Myers)
mcch = suffix tree (McCreight's algorithm with hash tables)
bmh = online search (Boyer-Moore-Horspool algorithm)

Comparison: Exact string matching online and offline

Experimental results: index construction plus ρn pattern searches for $\rho \in [0, 1]$





mamy = suffix array (implementation by Manber/Myers)
mcch = suffix tree (McCreight's algorithm with hash tables)
bmh = online search (Boyer-Moore-Horspool algorithm)
wotdlazy = suffix tree write-only top-down construction (lazy version)

Overview: Repeat analysis on a genomic scale

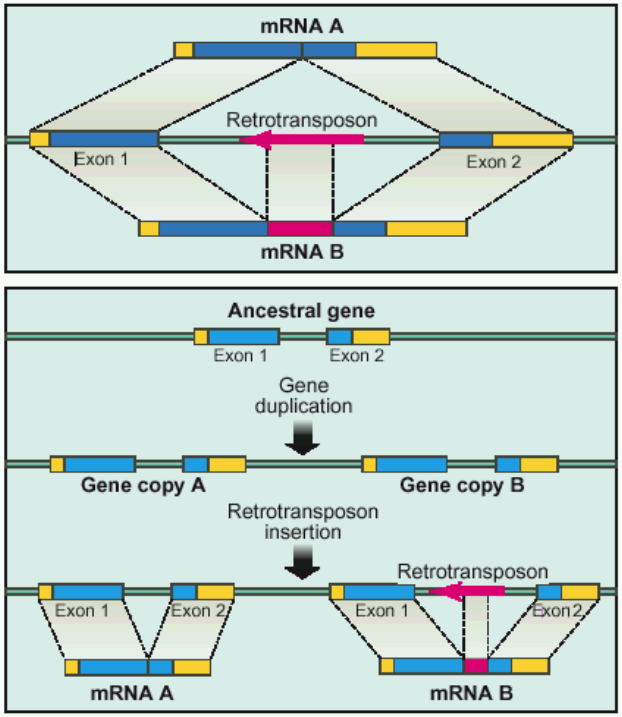
- Introduction
  - String pattern matching
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats
  - Degenerate repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

Repeats in biosequence analysis

- DNA of eukaryotes is highly repetitive.
  - 30–50% in human genome
  - 10% introduced by retroviruses?
- Repeat regions are rapidly changing *hot spots* in evolution.
- Vast literature on repetitive structures and their hypothesized functional and evolutionary roles: ALUs, SINEs, LINEs, microsatellites, minisatellites, ...
- Repeats are involved in several biological mechanisms, including genetically inherited diseases.
  - e.g. Huntington's disease
- Repeats tend to confuse sequence analysis programs and hence should be masked in a preprocessing step.

$\Rightarrow$ Repeats are very important when studying genomic DNA.

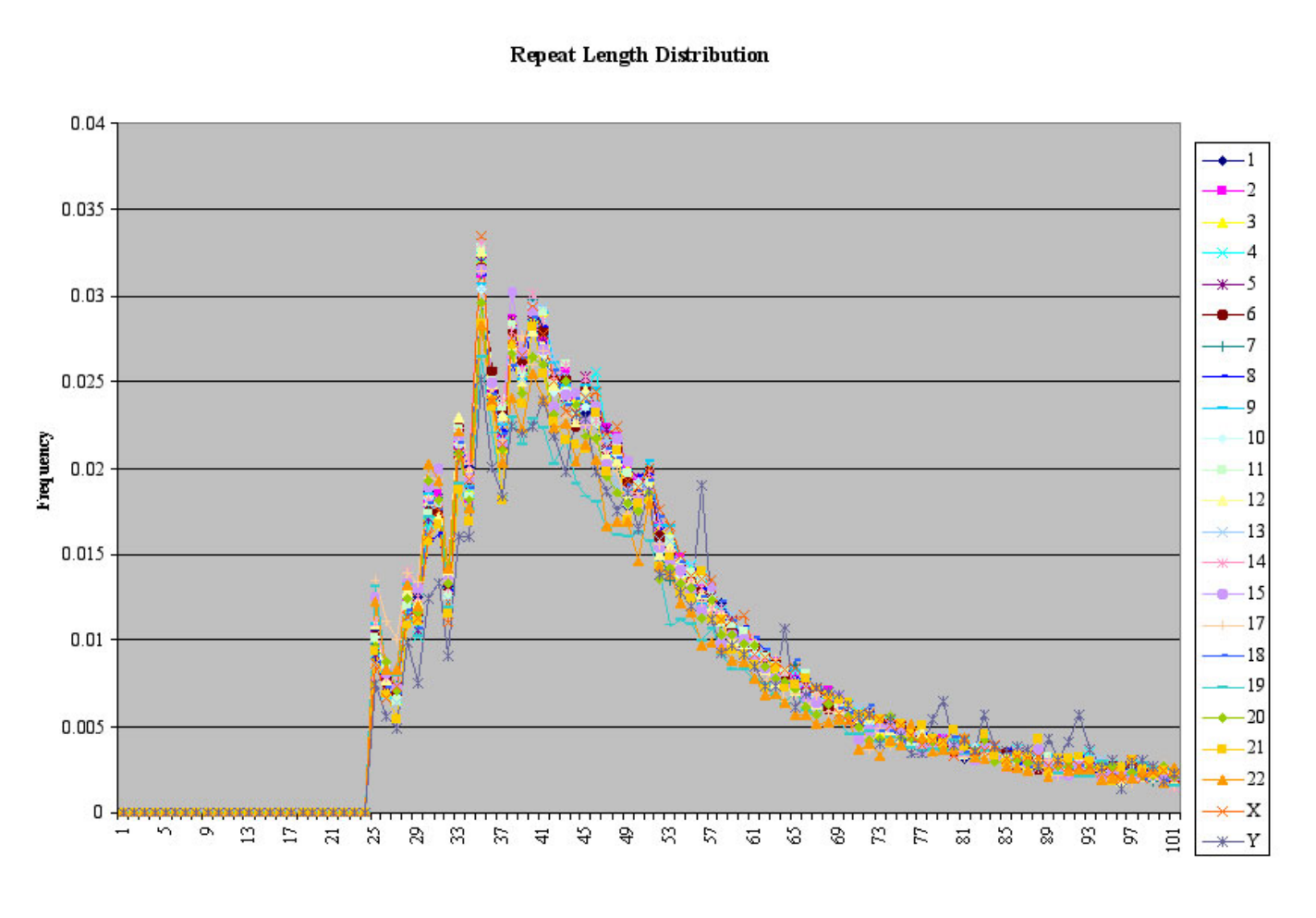
Effect of duplication



Junk DNA caught in the act. Two ways in which a repetitive DNA element, such as an Alu element, can be incorporated into the coding region of a gene without destroying the gene's function. (**Top**) A TE-cassette is inserted into the mRNA as an alternative exon. (**Bottom**) Insertion of a TE-cassette is preceded by a gene duplication. In both cases, the genome gains two forms of the mRNA transcript—one with and one without the TE-cassette.

Figure taken from: W. Makalowski, *Science* 300, 1246-1247, 2003.

Repeats in the human genome



(Human Genome Sequencing Center, Baylor College of Medicine, Houston, Texas)

Overview: Repeat analysis on a genomic scale

- Introduction
  - String pattern matching
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats
  - Degenerate repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

Overview: Repeat analysis on a genomic scale

- Introduction
  - String pattern matching
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats
  - Degenerate repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

Basic definitions

A pair of substrings $R = (S[i_1, j_1], S[i_2, j_2])$ is called a repeat.

$\longrightarrow$ *exact* repeat if $S[i_1, j_1] = S[i_2, j_2]$

Basic definitions

A pair of substrings $R = (S[i_1, j_1], S[i_2, j_2])$ is called a repeat.

$\longrightarrow$ *exact* repeat if $S[i_1, j_1] = S[i_2, j_2]$

Basic definitions

A pair of substrings $R = (S[i_1, j_1], S[i_2, j_2])$ is called a repeat.

$\longrightarrow$ *exact* repeat if $S[i_1, j_1] = S[i_2, j_2]$

$\longrightarrow$ *k-mismatch* repeat if there are k mismatches between $S[i_1, j_1]$ and $S[i_2, j_2]$

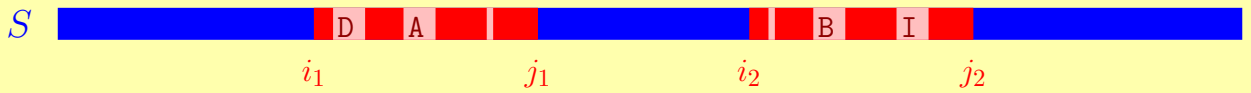
Basic definitions

A pair of substrings $R = (S[i_1, j_1], S[i_2, j_2])$ is called a repeat.

→ *exact* repeat if $S[i_1, j_1] = S[i_2, j_2]$

→ *k-mismatch* repeat if there are k mismatches between $S[i_1, j_1]$ and $S[i_2, j_2]$

→ *k-differences* repeat if there are k differences (mismatches, insertions, deletions) between $S[i_1, j_1]$ and $S[i_2, j_2]$



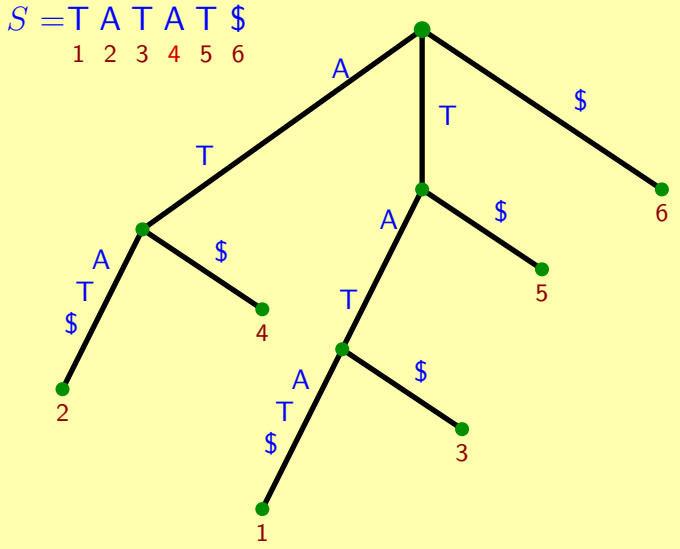
Finding exact repeats

Folklore: (see e.g. Gusfield, 1997)

- It is possible to find all pairs of repeated substrings (repeats) in S in linear time.

Idea:

- consider string S and its suffix tree $T(S)$.
- repeated substrings of S correspond to *internal locations* in $T(S)$.
- leaf numbers tell us positions where substrings occur.



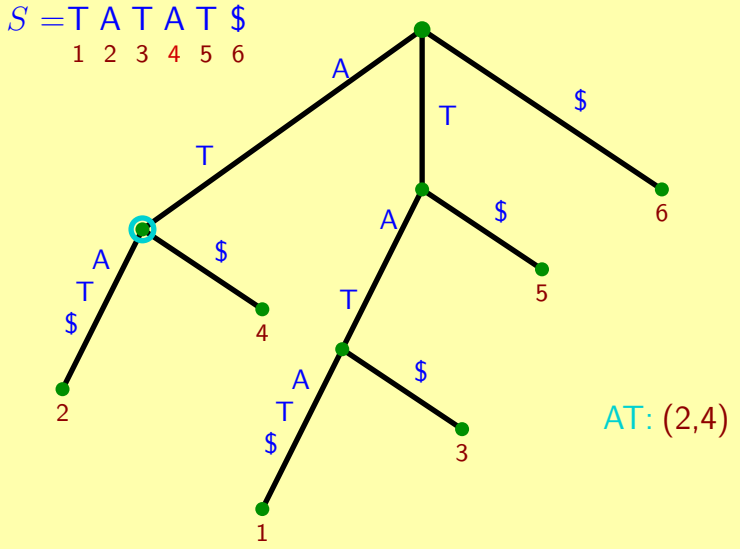
Finding exact repeats

Folklore: (see e.g. Gusfield, 1997)

- It is possible to find all pairs of repeated substrings (repeats) in S in linear time.

Idea:

- consider string S and its suffix tree $T(S)$.
- repeated substrings of S correspond to *internal locations* in $T(S)$.
- leaf numbers tell us positions where substrings occur.



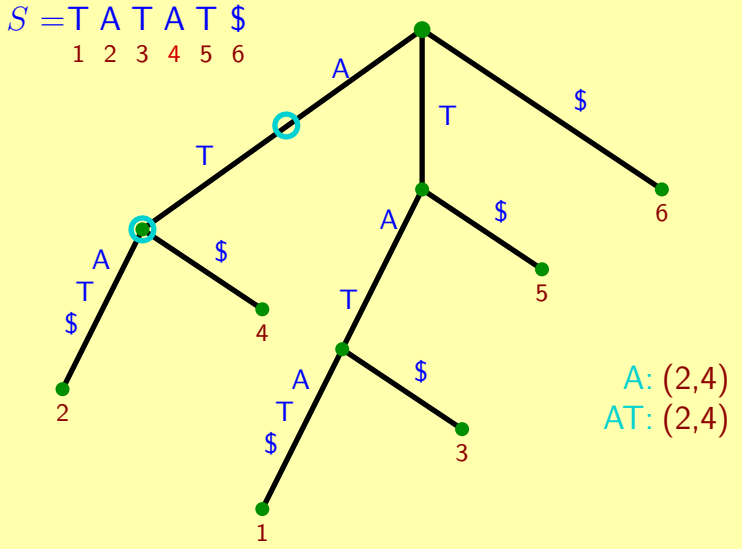
Finding exact repeats

Folklore: (see e.g. Gusfield, 1997)

- It is possible to find all pairs of repeated substrings (repeats) in S in linear time.

Idea:

- consider string S and its suffix tree $T(S)$.
- repeated substrings of S correspond to *internal locations* in $T(S)$.
- leaf numbers tell us positions where substrings occur.



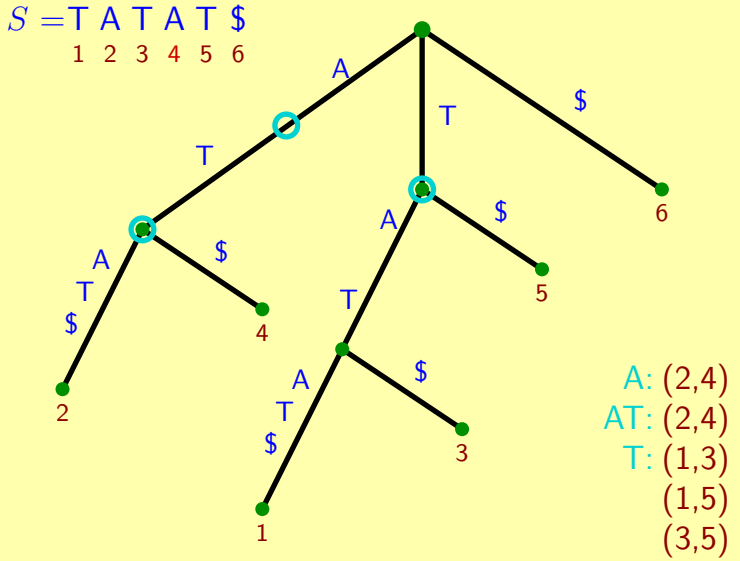
Finding exact repeats

Folklore: (see e.g. Gusfield, 1997)

- It is possible to find all pairs of repeated substrings (repeats) in S in linear time.

Idea:

- consider string S and its suffix tree $T(S)$.
- repeated substrings of S correspond to *internal locations* in $T(S)$.
- leaf numbers tell us positions where substrings occur.



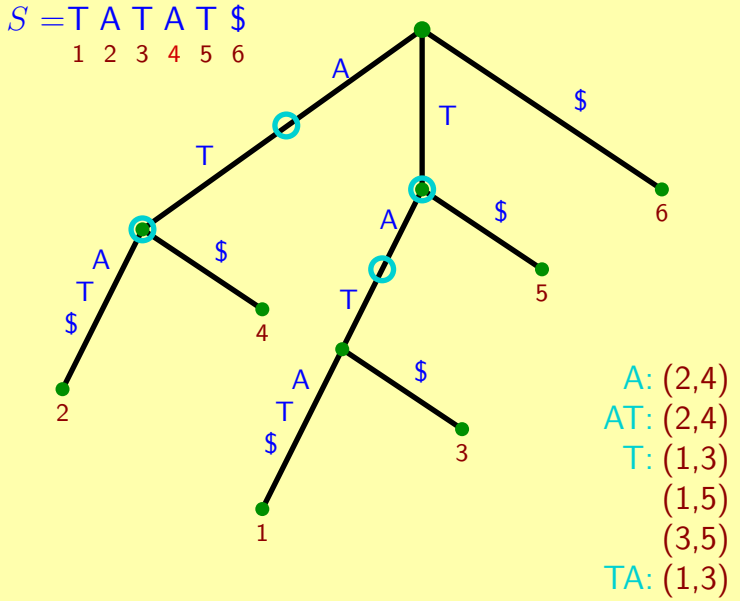
Finding exact repeats

Folklore: (see e.g. Gusfield, 1997)

- It is possible to find all pairs of repeated substrings (repeats) in S in linear time.

Idea:

- consider string S and its suffix tree $T(S)$.
- repeated substrings of S correspond to *internal locations* in $T(S)$.
- leaf numbers tell us positions where substrings occur.



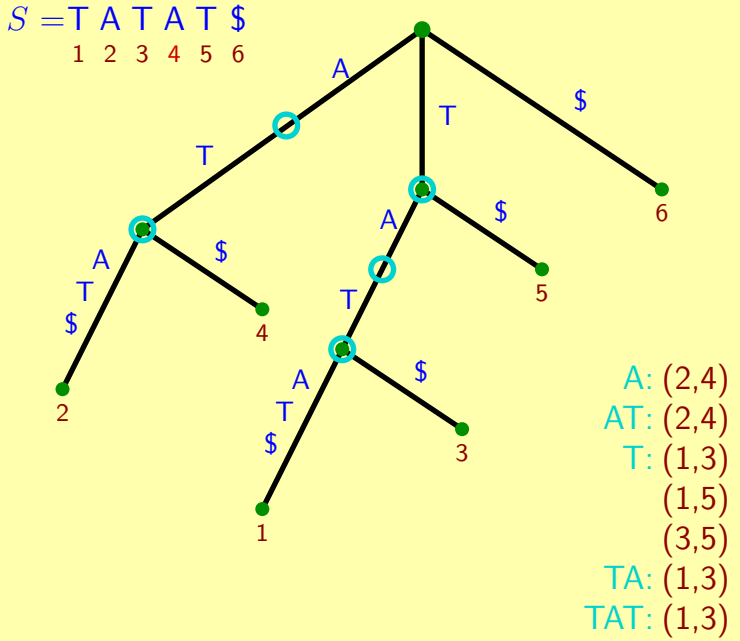
Finding exact repeats

Folklore: (see e.g. Gusfield, 1997)

- It is possible to find all pairs of repeated substrings (repeats) in S in linear time.

Idea:

- consider string S and its suffix tree $T(S)$.
- repeated substrings of S correspond to *internal locations* in $T(S)$.
- leaf numbers tell us positions where substrings occur.



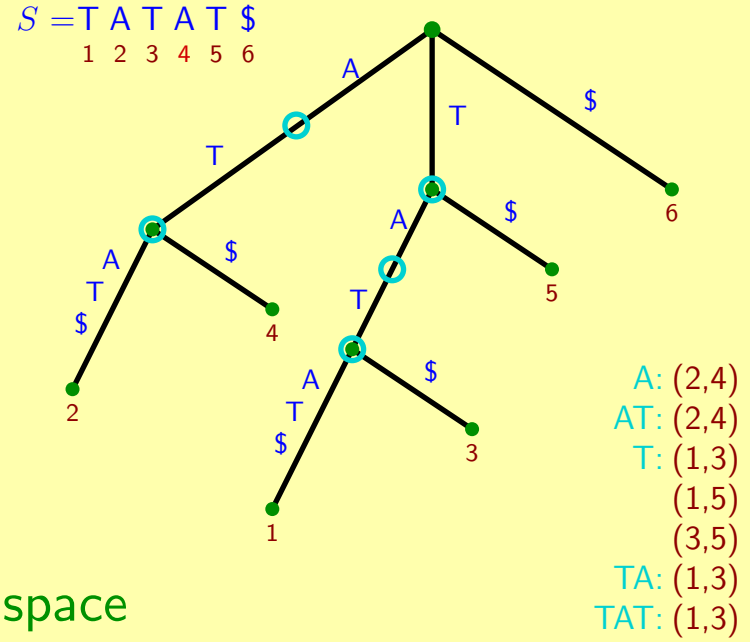
Finding exact repeats

Folklore: (see e.g. Gusfield, 1997)

- It is possible to find all pairs of repeated substrings (repeats) in S in linear time.

Idea:

- consider string S and its suffix tree $T(S)$.
- repeated substrings of S correspond to *internal locations* in $T(S)$.
- leaf numbers tell us positions where substrings occur.

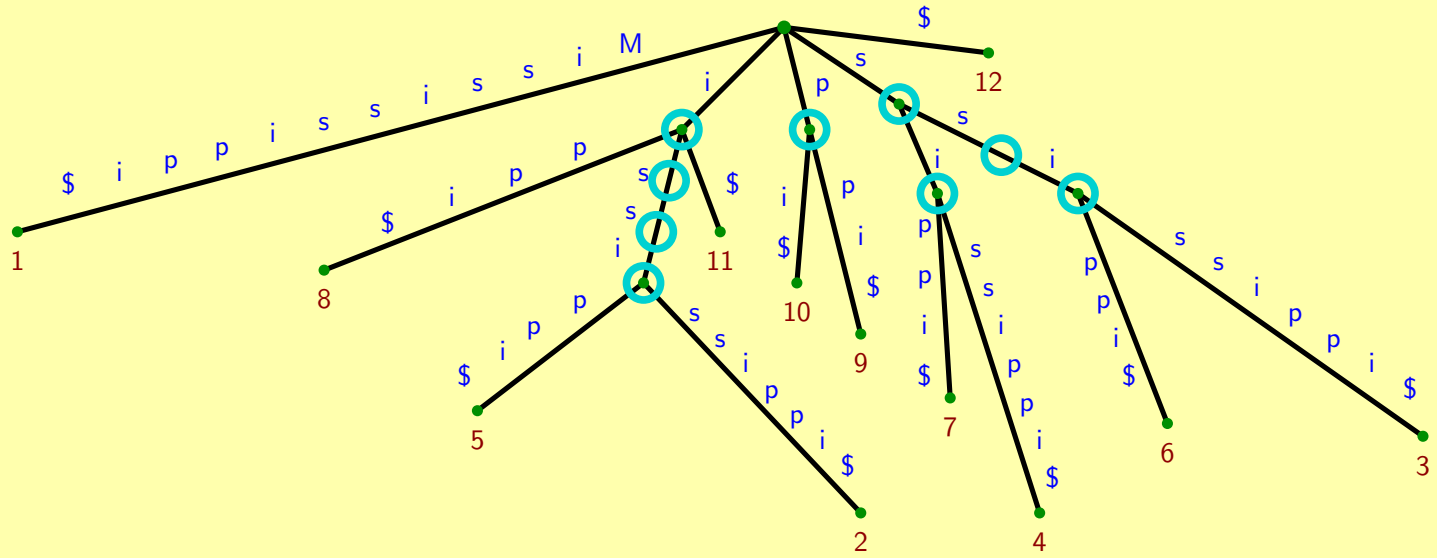


Analysis: $\mathcal{O}(n+z)$ time with $z = |\text{output}|$, $\mathcal{O}(n)$ space

A larger example

$S =$ M i s s i s s i p p i $

1 2 3 4 5 6 7 8 9 10 11 12



i: (8,5)	is: (5,2)	p: (10,9)	s: (7,4)	si: (7,4)
(8,2)			(7,6)	
(8,11)	iss: (5,2)		(7,3)	ss: (6,3)
(5,2)			(4,6)	
(5,11)	issi: (5,2)		(4,3)	ssi: (6,3)
(2,11)			(6,3)	

Finding *maximal* exact repeats

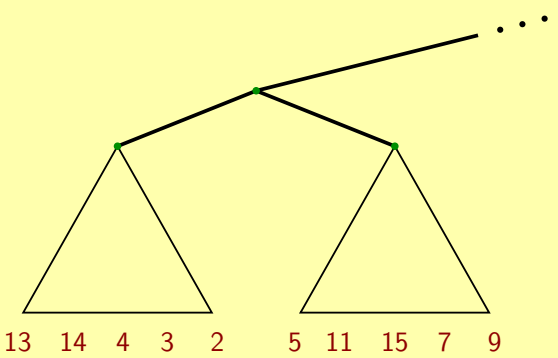
Finding *maximal* exact repeats

Finding *maximal* exact repeats

Finding *maximal* exact repeats

Idea:

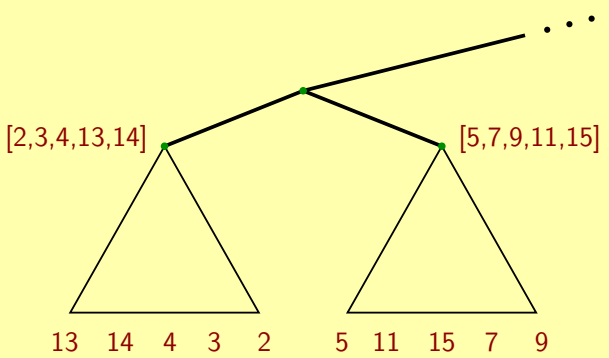
- For right-maximality ($X \neq Y$)
  - consider only internal nodes of $T(S)$
  - report only pairs of leaves from different subtrees



Finding *maximal* exact repeats

Idea:

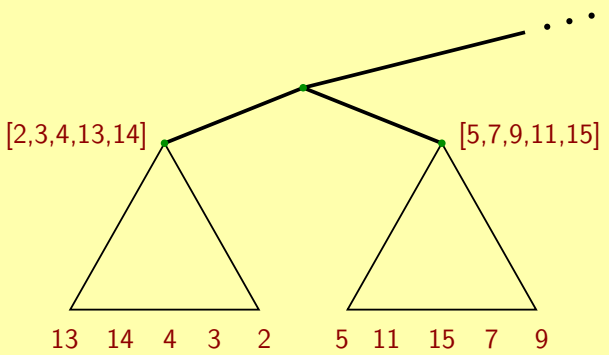
- For right-maximality (X $\neq$ Y)
  - consider only internal nodes of $T(S)$
  - report only pairs of leaves from different subtrees (or from different leaf-lists)



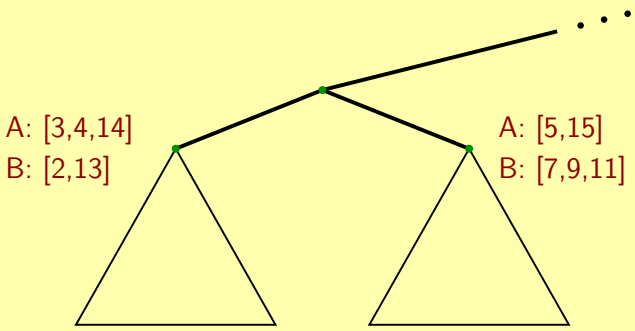
Finding *maximal* exact repeats

Idea:

- For right-maximality (X $\neq$ Y)
  - consider only internal nodes of $T(S)$
  - report only pairs of leaves from different subtrees (or from different leaf-lists)



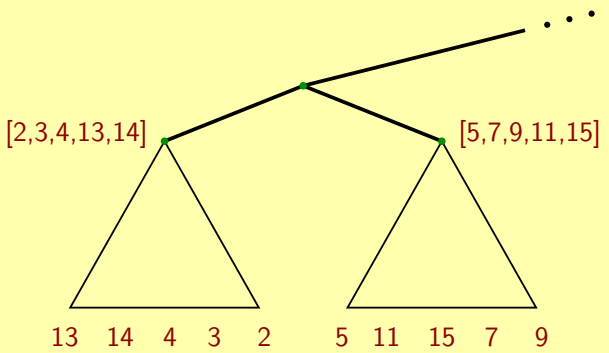
- For left-maximality (A $\neq$ B)
  - keep lists for the different left-characters
  - report only pairs from different lists



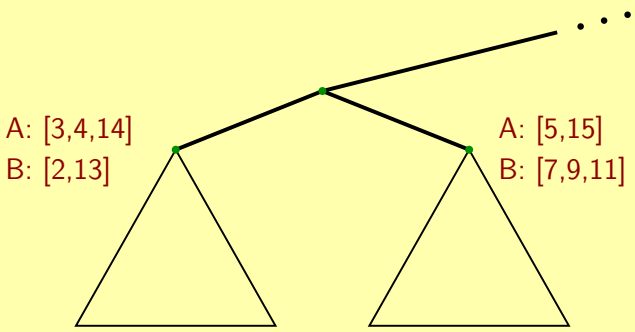
Finding *maximal* exact repeats

Idea:

- For right-maximality (X $\neq$ Y)
  - consider only internal nodes of $T(S)$
  - report only pairs of leaves from different subtrees (or from different leaf-lists)

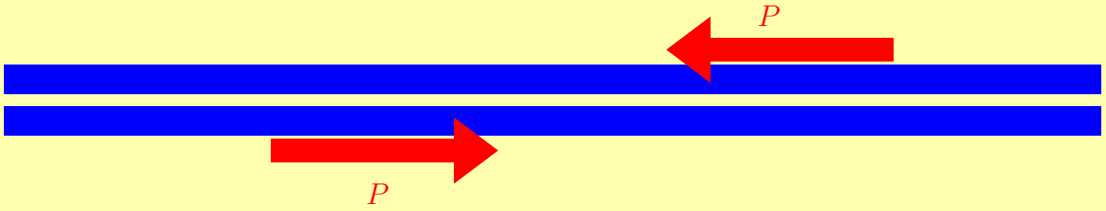


- For left-maximality (A $\neq$ B)
  - keep lists for the different left-characters
  - report only pairs from different lists

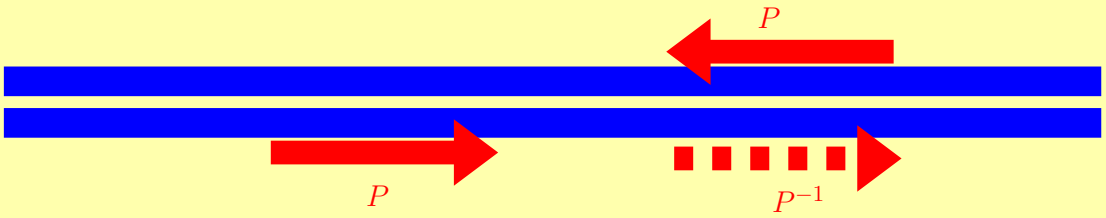


Analysis: $\mathcal{O}(n+z)$ time with $z = |\text{output}|$, $\mathcal{O}(n)$ space

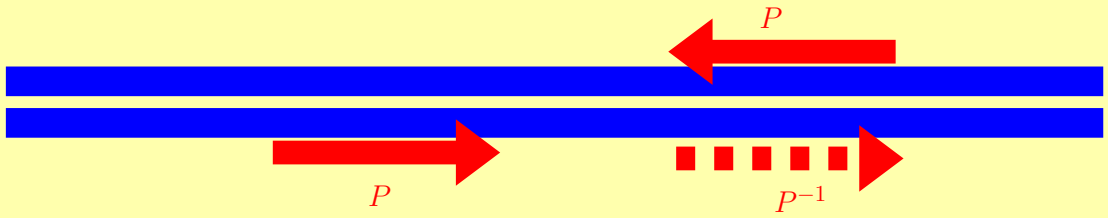
Variation: Palindromic repeats



Variation: Palindromic repeats

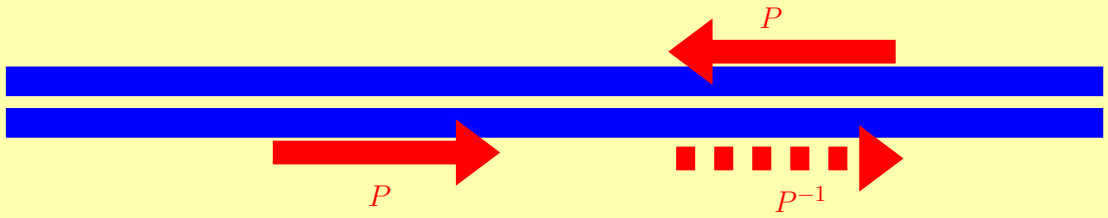


Variation: Palindromic repeats

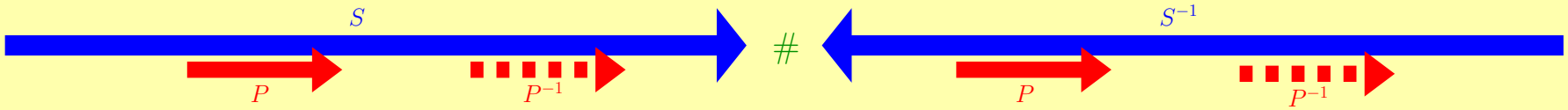


- One repeat instance must be reverse Watson/Crick complement P^{-1}.
- Essentially same problem as computing direct repeats.
- Instead of S use $S\#S^{-1}$ (where S^{-1} is the reverse complement of S).

Variation: Palindromic repeats



- One repeat instance must be reverse Watson/Crick complement P^{-1}.
- Essentially same problem as computing direct repeats.
- Instead of S use $S\#S^{-1}$ (where S^{-1} is the reverse complement of S).



- $\#$ is a unique separator symbol.
- One of the duplicates must be in S and the other in S^{-1}.
- Calculate position in S^{-1} relative to the beginning of S.

Overview: Repeat analysis on a genomic scale

- Introduction
  - String pattern matching
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats
  - Degenerate repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

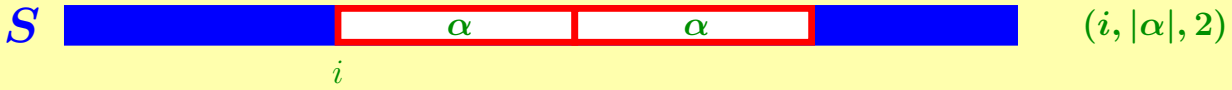
Tandem repeats: Definitions

- tandem repeat (square)

Tandem repeats: Definitions

- tandem repeat (square)

- occurrence of a tandem repeat

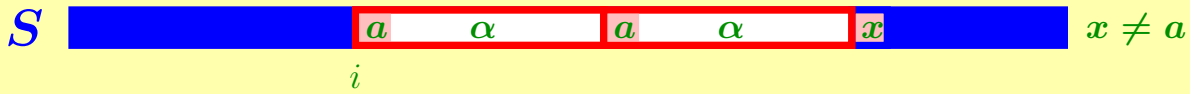


Tandem repeats: Definitions

- tandem repeat (square)

- occurrence of a tandem repeat

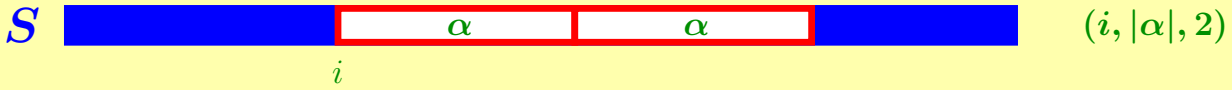
- (right-) branching occurrence of a tandem repeat



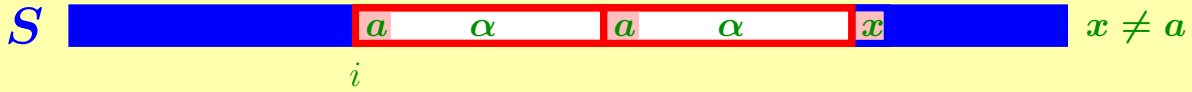
Tandem repeats: Definitions

- tandem repeat (square)

- occurrence of a tandem repeat



- (right-) branching occurrence of a tandem repeat



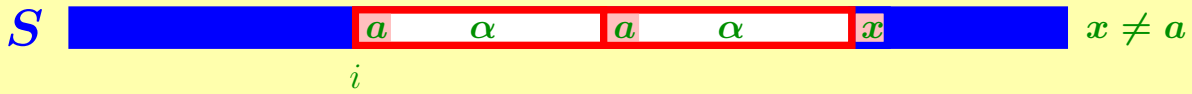
- a string w is primitive if and only if $w = u^k$ implies $k = 1$

Tandem repeats: Definitions

- tandem repeat (square)

- occurrence of a tandem repeat

- (right-) branching occurrence of a tandem repeat



- a string w is primitive if and only if $w = u^k$ implies $k = 1$
- a tandem repeat $\alpha\alpha$ is primitive if and only if α is primitive

Finding tandem repeats: Overview

A. Find all ccurrences of tandem repeats in a string.

- Main/Lorentz, 1979/1984
- Landau/Schmidt, 1993

B. Find all occurrences of *primitive* tandem repeats in a string.

- Crochemore, 1981
- Apostolico/Preparata, 1983

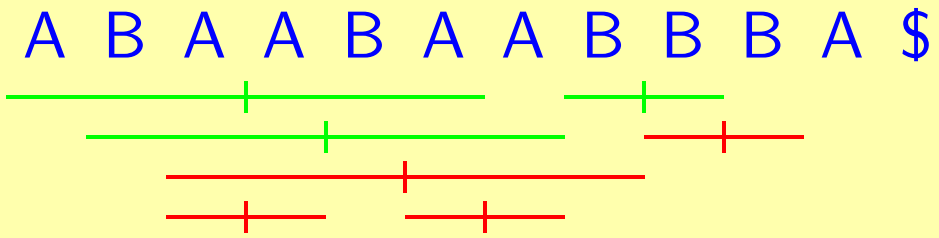
C. Find all occurrences of primitive tandem *arrays* in a string.

Here:
Simple and flexible detection of all of these in optimal time using a suffix tree.
(joint work with Dan Gusfield)

Basic observation

Lemma:
Any non-branching occurrence $(i, l, 2)$ of a tandem repeat is the left-rotation of another tandem repeat $(i + 1, l, 2)$, starting one position to its right.

Example:



Suffix trees and tandem repeats

Lemma: (folklore)
Consider two positions i and j of S, $1 \leq i < j \leq n$, let $l = j - i$. Then the following assertions are equivalent:

(a) $(i, l, 2)$ is an occurrence of a tandem repeat;

(b) i and j occur in the same leaf-list of some node v in $T(S)$ with depth $D(v) \geq l$.

Example:

A	B	A	A	B	A	A	B	B	B	A	$
1	2	3	4	5	6	7	8	9	10	11	12

Suffix trees and tandem repeats

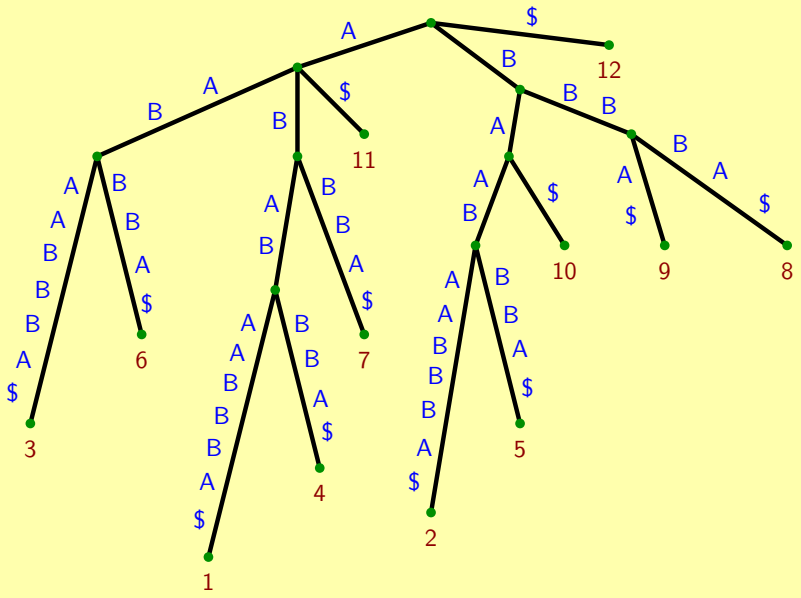
Lemma: (folklore)
Consider two positions i and j of S, $1 \leq i < j \leq n$, let $l = j - i$. Then the following assertions are equivalent:

(a) $(i, l, 2)$ is an occurrence of a tandem repeat;

(b) i and j occur in the same leaf-list of some node v in $T(S)$ with depth $D(v) \geq l$.

Example:

A	B	A	A	B	A	A	B	B	B	A	$
1	2	3	4	5	6	7	8	9	10	11	12



Suffix trees and tandem repeats

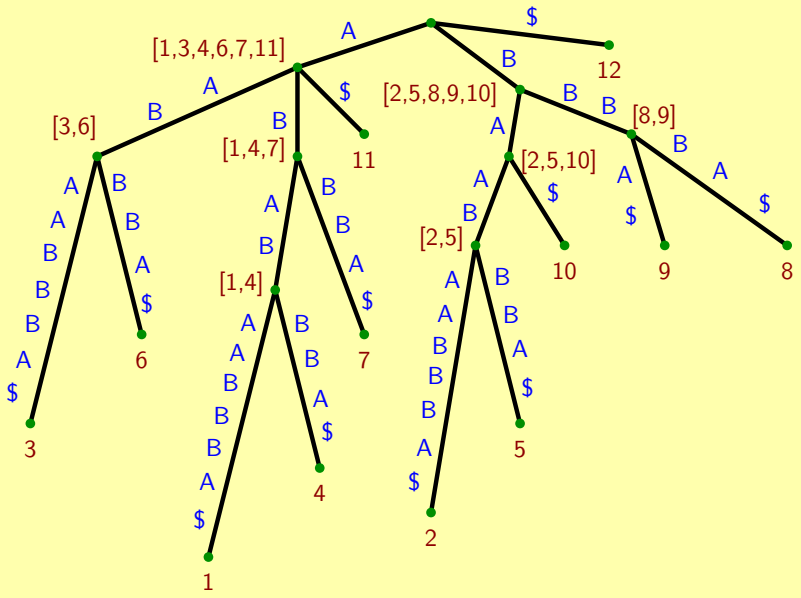
Lemma: (folklore)
Consider two positions i and j of S, $1 \leq i < j \leq n$, let $l = j - i$. Then the following assertions are equivalent:

(a) $(i, l, 2)$ is an occurrence of a tandem repeat;

(b) i and j occur in the same leaf-list of some node v in $T(S)$ with depth $D(v) \geq l$.

Example:

A	B	A	A	B	A	A	B	B	B	A	$
1	2	3	4	5	6	7	8	9	10	11	12



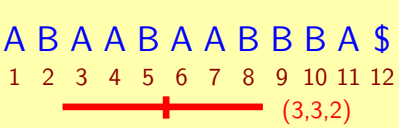
Suffix trees and tandem repeats

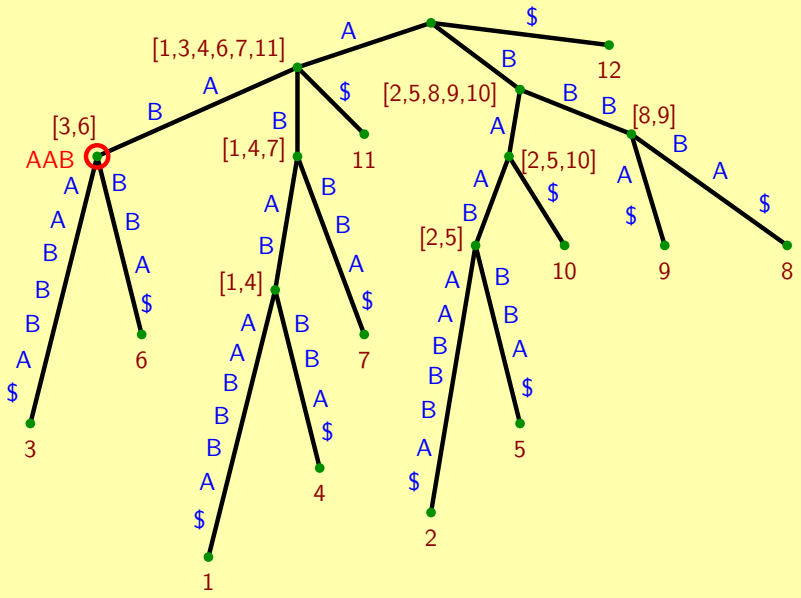
Lemma: (folklore)
Consider two positions i and j of S, $1 \leq i < j \leq n$, let $l = j - i$. Then the following assertions are equivalent:

(a) $(i, l, 2)$ is an occurrence of a tandem repeat;

(b) i and j occur in the same leaf-list of some node v in $T(S)$ with depth $D(v) \geq l$.

Example:





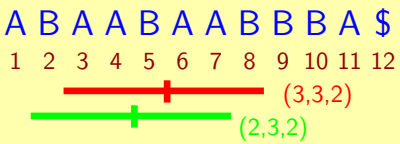
Suffix trees and tandem repeats

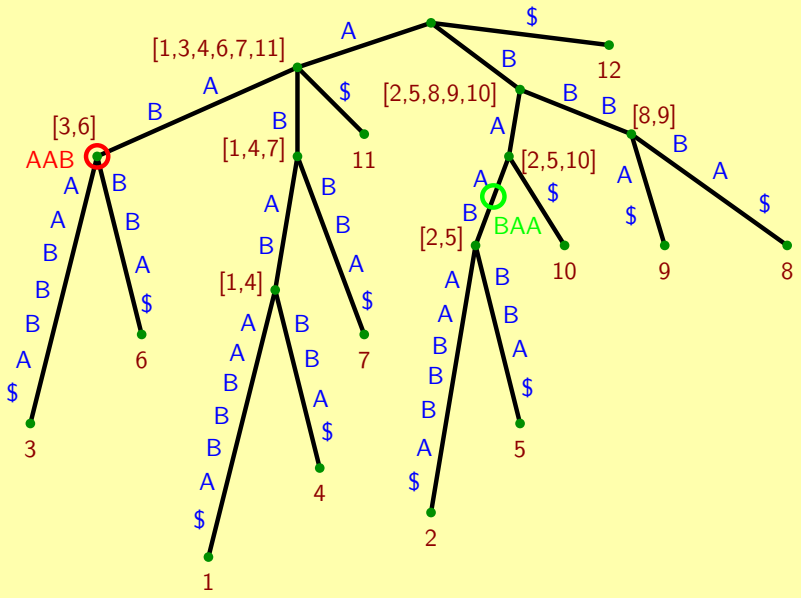
Lemma: (folklore)
Consider two positions i and j of S, $1 \leq i < j \leq n$, let $l = j - i$. Then the following assertions are equivalent:

(a) $(i, l, 2)$ is an occurrence of a tandem repeat;

(b) i and j occur in the same leaf-list of some node v in $T(S)$ with depth $D(v) \geq l$.

Example:





Suffix trees and *branching* tandem repeats

Lemma:
Consider two positions i and j of S, $1 \leq i < j \leq n$, let $l = j - i$. Then the following assertions are equivalent:

(a) $(i, l, 2)$ is an occurrence of a branching tandem repeat;

(b) i and j occur in the same leaf-list of some node v in $T(S)$ with depth $D(v) = l$, but do not appear in the same leaf-list of any node with depth greater than l.

Suffix trees and *branching* tandem repeats

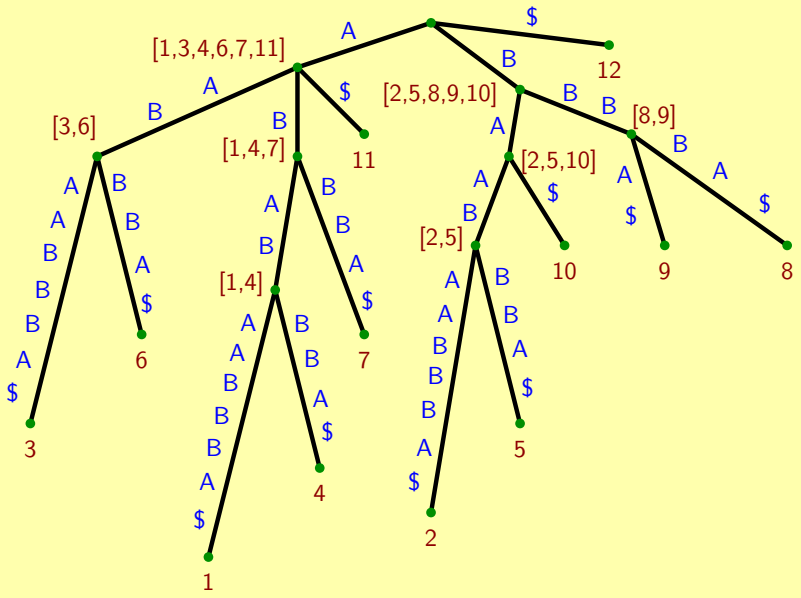
Lemma:
Consider two positions i and j of S, $1 \leq i < j \leq n$, let $l = j - i$. Then the following assertions are equivalent:

(a) $(i, l, 2)$ is an occurrence of a branching tandem repeat;

(b) i and j occur in the same leaf-list of some node v in $T(S)$ with depth $D(v) = l$, but do not appear in the same leaf-list of any node with depth greater than l.

Example:

A	B	A	A	B	A	A	B	B	B	A	$
1	2	3	4	5	6	7	8	9	10	11	12



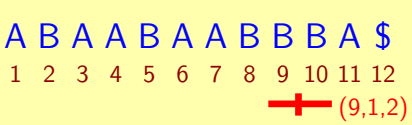
Suffix trees and *branching* tandem repeats

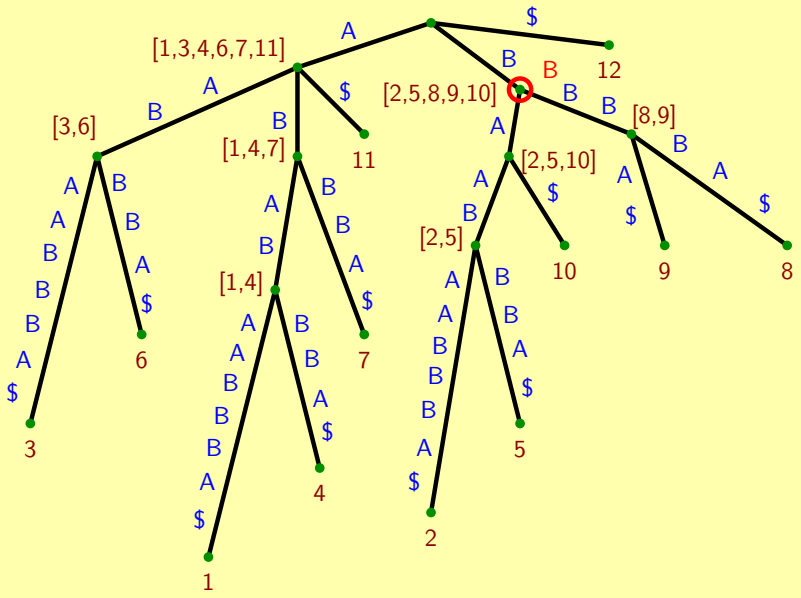
Lemma:
Consider two positions i and j of S, $1 \leq i < j \leq n$, let $l = j - i$. Then the following assertions are equivalent:

(a) $(i, l, 2)$ is an occurrence of a branching tandem repeat;

(b) i and j occur in the same leaf-list of some node v in $T(S)$ with depth $D(v) = l$, but do not appear in the same leaf-list of any node with depth greater than l.

Example:





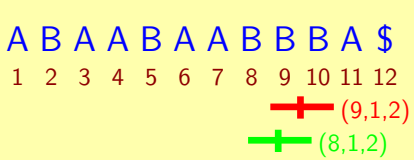
Suffix trees and *branching* tandem repeats

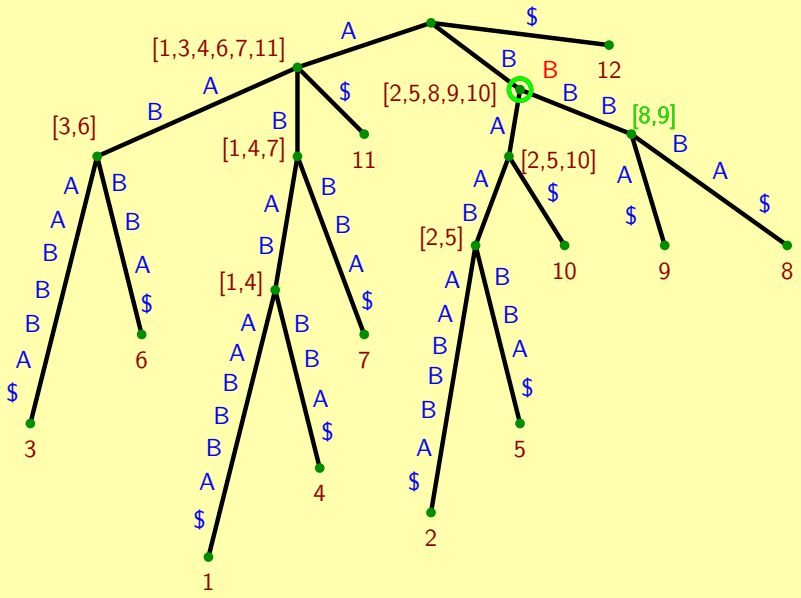
Lemma:
Consider two positions i and j of S, $1 \leq i < j \leq n$, let $l = j - i$. Then the following assertions are equivalent:

(a) $(i, l, 2)$ is an occurrence of a branching tandem repeat;

(b) i and j occur in the same leaf-list of some node v in $T(S)$ with depth $D(v) = l$, but do not appear in the same leaf-list of any node with depth greater than l.

Example:





Basic algorithm

Idea:
For each node v of $T(S)$, test if $\alpha\alpha = L(v)L(v)$ is a branching tandem repeat.

Algorithm:

All nodes of $T(S)$ begin unmarked.
Step 1 is repeated until all nodes are marked.

1. Select an unmarked internal node v.
   Mark v and execute steps 2a and 2b for node v.

2a. Collect the leaf-list $LL(v)$.

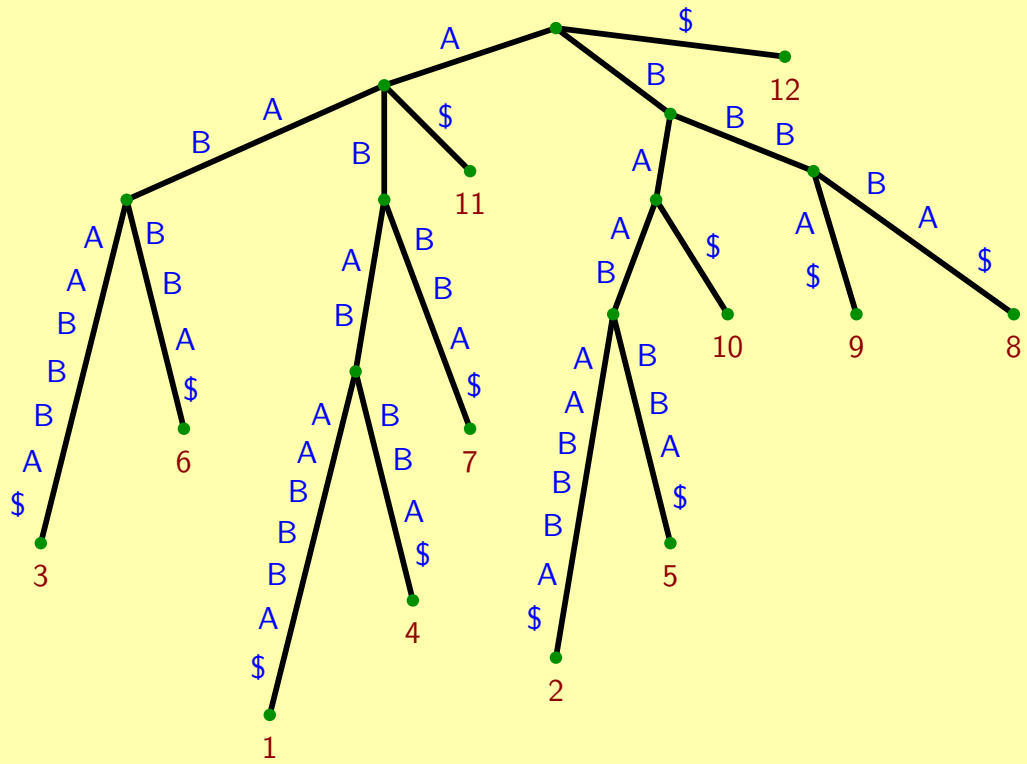
2b. For each leaf i in $LL(v)$, test whether the leaf $j = i + D(v)$ is in $LL(v)$.
   If so, test whether $S[i] \neq S[i + 2D(v)]$. There is a branching tandem repeat of length $2D(v)$ starting at position i if and only if both tests return true.

Analysis: $\mathcal{O}(n^2)$ time, $\mathcal{O}(n)$ space

Testing in constant time

Depth-first numbering and look-up table:

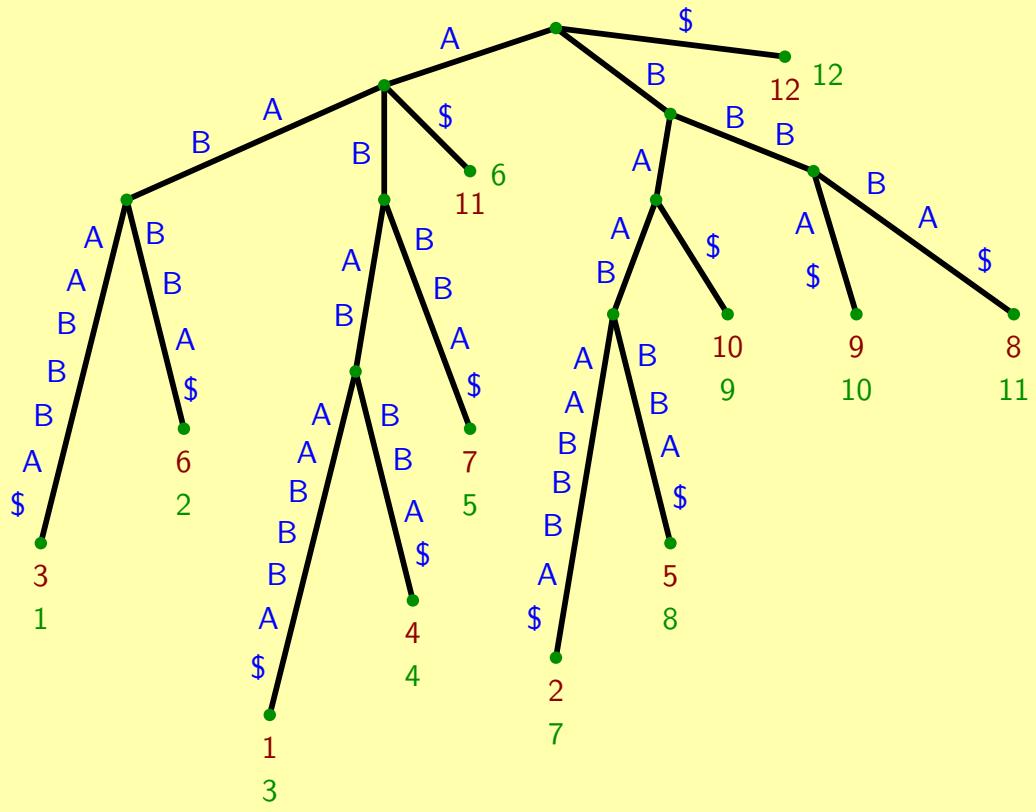
A	B	A	A	B	A	A	B	B	B	A	$
1	2	3	4	5	6	7	8	9	10	11	12



Testing in constant time

Depth-first numbering and look-up table:

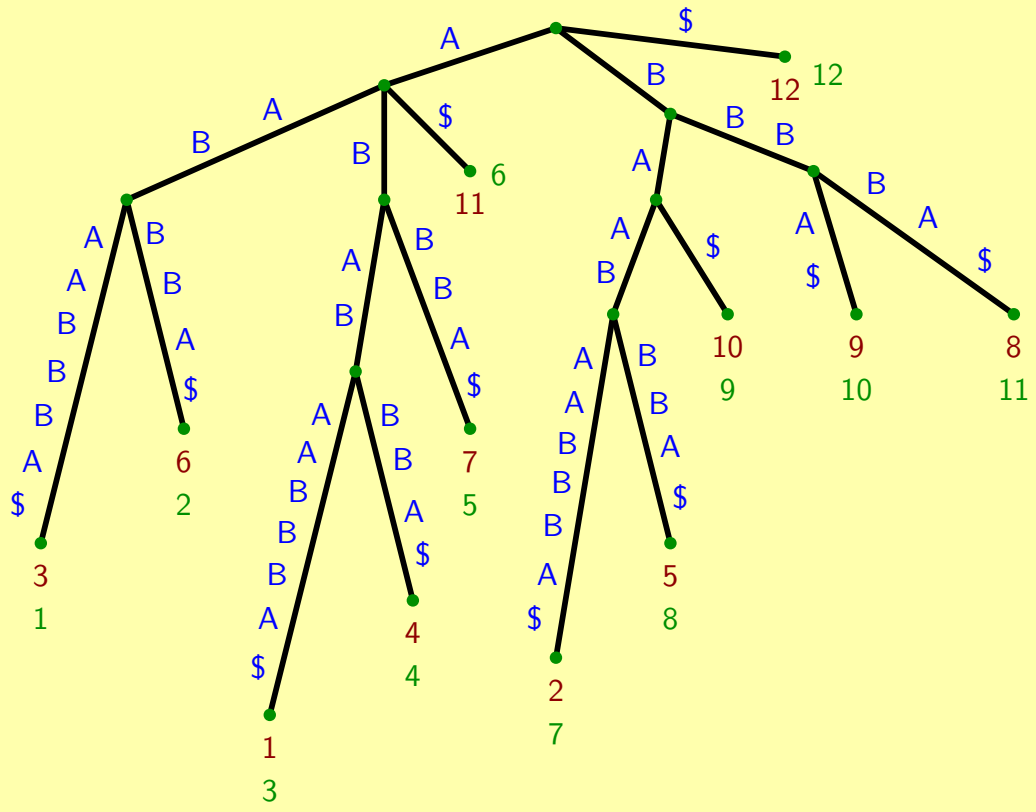
A	B	A	A	B	A	A	B	B	B	A	$
1	2	3	4	5	6	7	8	9	10	11	12



Testing in constant time

Depth-first numbering and look-up table:

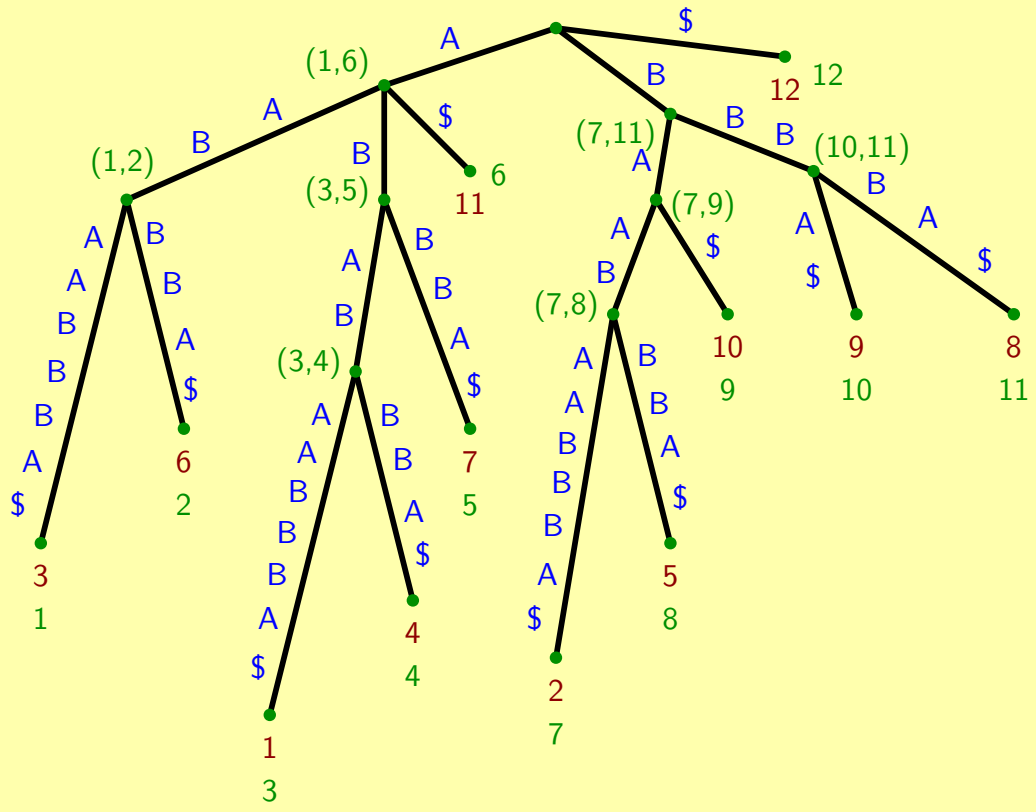
A	B	A	A	B	A	A	B	B	B	A	$
1	2	3	4	5	6	7	8	9	10	11	12
3	7	1	4	8	2	5	11	10	9	6	12



Testing in constant time

Depth-first numbering and look-up table:

A	B	A	A	B	A	A	B	B	B	A	$
1	2	3	4	5	6	7	8	9	10	11	12
3	7	1	4	8	2	5	11	10	9	6	12



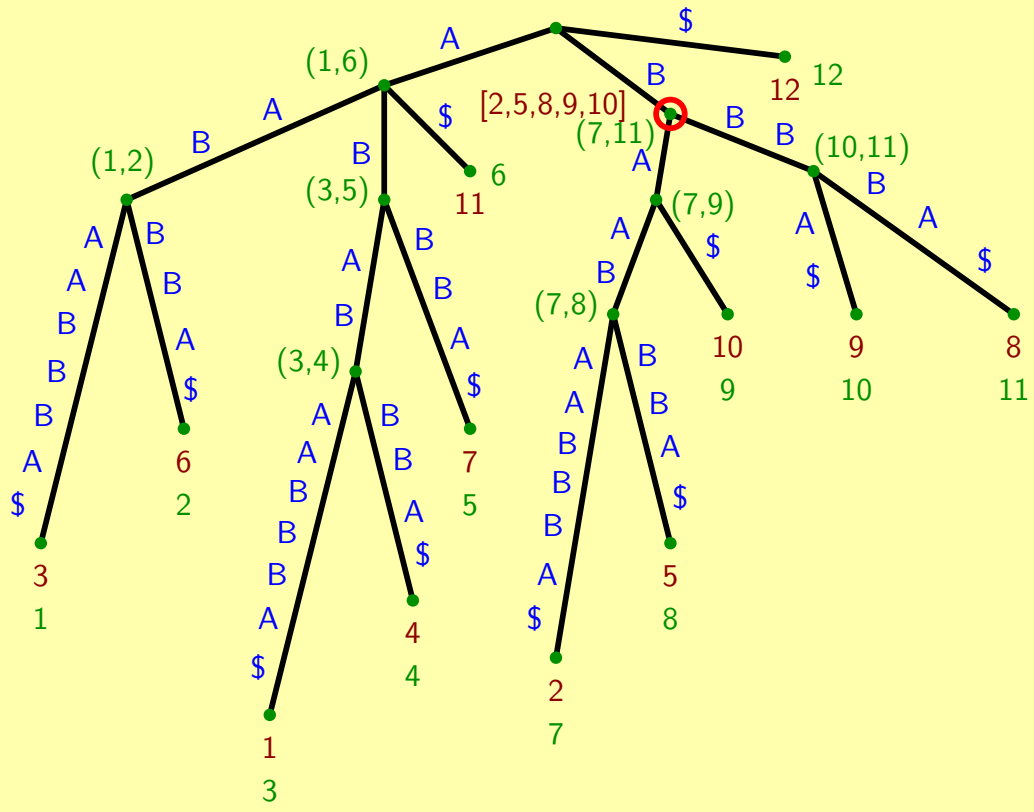
Testing in constant time

Depth-first numbering and look-up table:

A	B	A	A	B	A	A	B	B	B	A	$
1	2	3	4	5	6	7	8	9	10	11	12
3	7	1	4	8	2	5	11	10	9	6	12

B: 8 $\xrightarrow{D=1}$ 9? $\xrightarrow{table}$ $10 \in (7, 11)$

but: $(8, 9)$ not branching



Testing in constant time

Depth-first numbering and look-up table:

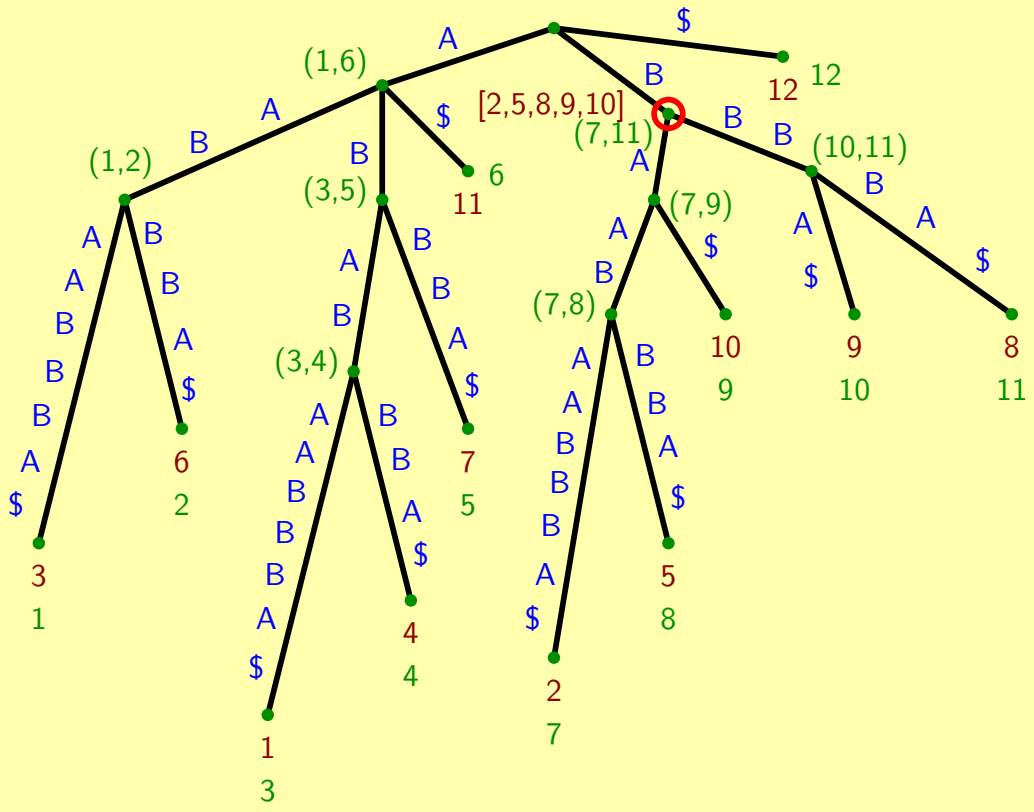
A	B	A	A	B	A	A	B	B	B	A	$
1	2	3	4	5	6	7	8	9	10	11	12
3	7	1	4	8	2	5	11	10	9	6	12

B: 8 $\xrightarrow{D=1}$ 9? $\xrightarrow{table}$ $10 \in (7, 11)$

but: $(8, 9)$ not branching

9 $\xrightarrow{D=1}$ 10? $\xrightarrow{table}$ $9 \in (7, 11)$

and: $(9, 10)$ is branching



Testing in constant time

Depth-first numbering and look-up table:

A	B	A	A	B	A	A	B	B	B	A	$
1	2	3	4	5	6	7	8	9	10	11	12
3	7	1	4	8	2	5	11	10	9	6	12

B: 8 $\xrightarrow{D=1}$ 9? $\xrightarrow{table}$ $10 \in (7, 11)$

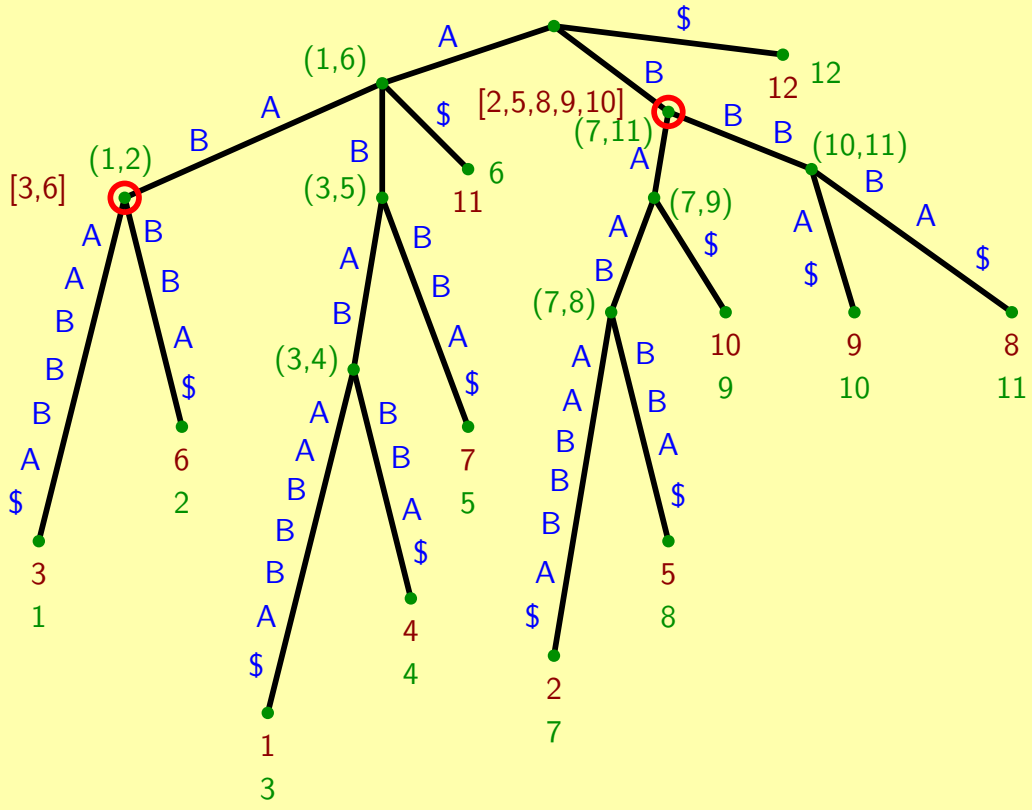
but: $(8, 9)$ not branching

9 $\xrightarrow{D=1}$ 10? $\xrightarrow{table}$ $9 \in (7, 11)$

and: $(9, 10)$ is branching

AAB: 3 $\xrightarrow{D=3}$ 6? $\xrightarrow{table}$ $2 \in (1, 2)$

and: $(3, 6)$ is branching



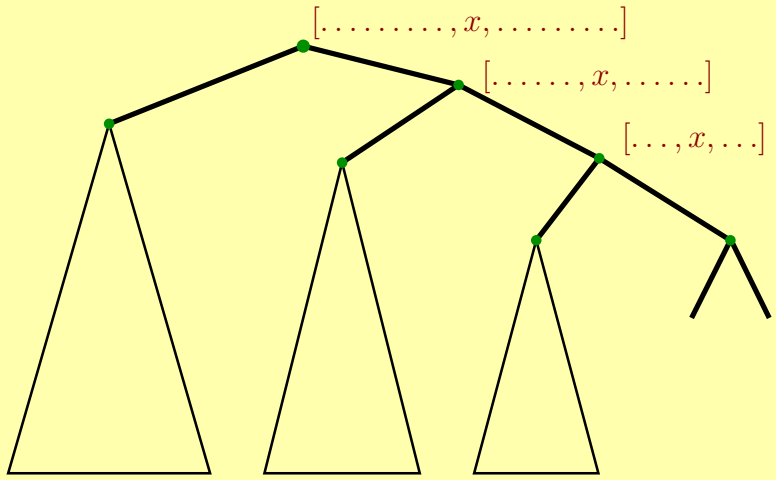
Speedup of the basic algorithm

Definitions

- For each node v, v' denotes the child of v with the largest leaf-list.
- $LL'(v)$ denotes $LL(v) - LL(v')$.

The "Smaller Half" Trick

It is well known that $\sum_v |LL'(v)| \leq n \log_2 n$.



Any value x can be
in at most $\log_2 n$ leaf-lists LL'.

Optimized basic algorithm

Algorithm:

All nodes of $T(S)$ begin unmarked.
Step 1 is repeated until all nodes are marked.

1. Select an unmarked internal node v.
   Mark v and execute steps 2a, 2b and 2c for node v.

2a. Collect the list $LL'(v)$ for v.

2b. For each leaf i in $LL'(v)$, test whether leaf $j = i + D(v)$ is in $LL(v)$, the leaf-list of v. If so, test whether $S[i] \neq S[i + 2D(v)]$. There is a branching tandem repeat of length $2D(v)$ starting at position i if and only if both tests return true.

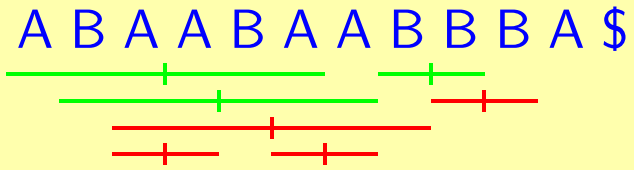
2c. Do the same test for each leaf j in $LL'(v)$, and $i = j - D(v)$.

Analysis: $\mathcal{O}(n \log n)$ time, $\mathcal{O}(n)$ space

Putting things together

Finding all tandem repeats

Starting at each of the branching occurrences, do a series of consecutive left-rotations to find all tandem repeats.



Analysis: $\mathcal{O}(n \log n + |\text{output}|)$ time, no additional space

Tandem repeats in linear time!

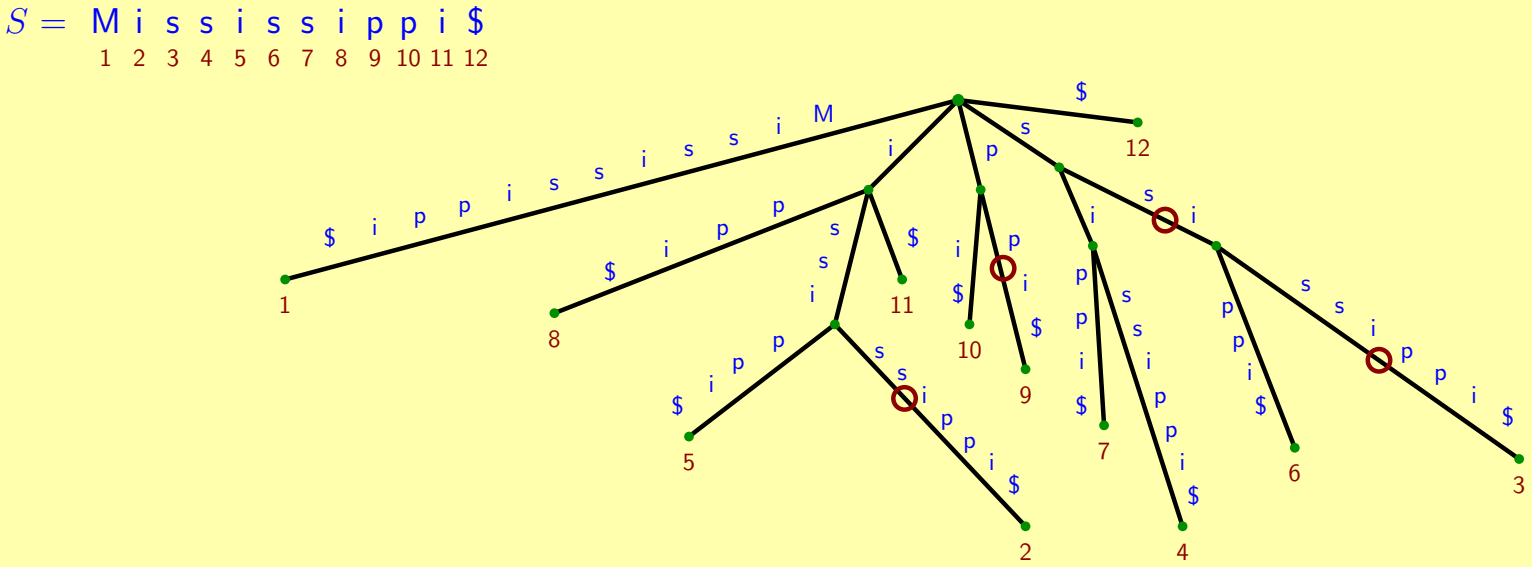
(joint work with D. Gusfield)

Fraenkel & Simpson, 1998:

- The vocabulary of all tandem repeats in S has only $\mathcal{O}(n)$ elements.

Idea:

- Mark in $T(S)$ all end points of tandem repeats.

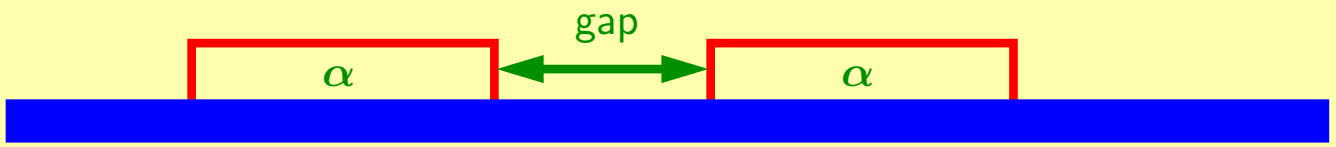


Analysis: $\mathcal{O}(n + |\text{output}|)$ time, $\mathcal{O}(n)$ space

Repeats with bounded gap

(joint work with G. S. Brodal, R. B. Lyngø, C. N. S. Pedersen)

Sometimes one wishes to allow between the copies of a repeat a gap of (upper and/or lower) bounded size.



Idea:

- Traverse the suffix tree bottom-up.
- At each vertex v collect the leaf-list $LL'(v)$.
- Output only pairs that have the required distance.

Analysis: $\mathcal{O}(n \log n + |\text{output}|)$ resp. $\mathcal{O}(n + |\text{output}|)$ time, $\mathcal{O}(n)$ space.

Overview: Repeat analysis on a genomic scale

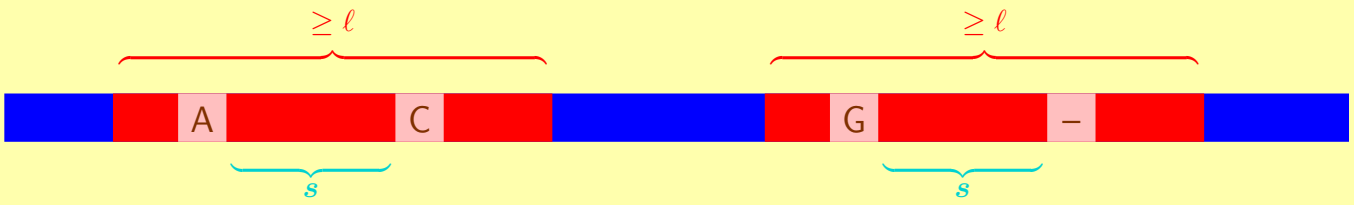
- Introduction
  - String pattern matching
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats
  - Degenerate repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

Finding *degenerate* repeats

(joint work with R. Giegerich, S. Kurtz, E. Ohlebisch, C. Schleiermacher)

Often, repats in genomic DNA are degenerate, i.e. at some positions more than one base is possible.

Idea: Filter method (seed and extend)



$s \geq \left\lfloor \frac{\ell}{k+1} \right\rfloor$

Algorithm:

1. Search for local exact repeats (seeds).
2. Extend the seeds while allowing up to k errors.
3. If extension is long enough, output repeat.

Analysis: $\mathcal{O}(n + \zeta k^3)$ time with $E(\zeta) = O\left(n^2/4^s\right)$, s minimal seed length.

Extension for maximal k-mismatch repeats

Simple extension and length test:

Extension for maximal k-mismatch repeats

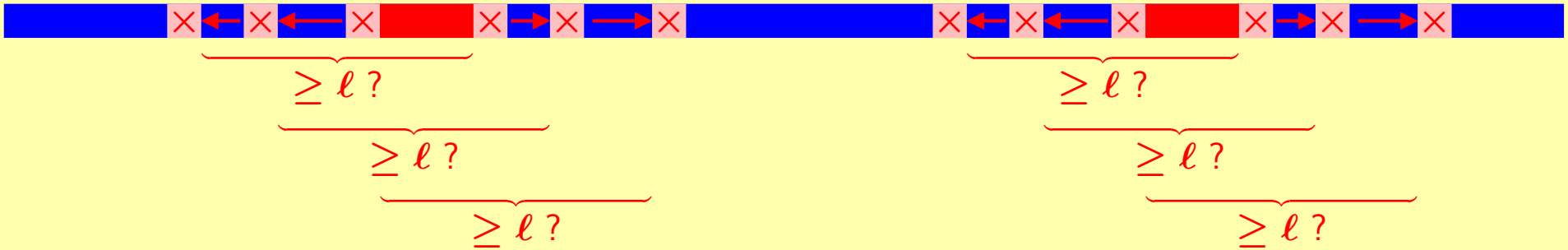
Simple extension and length test:

Extension for maximal k-mismatch repeats

Simple extension and length test:

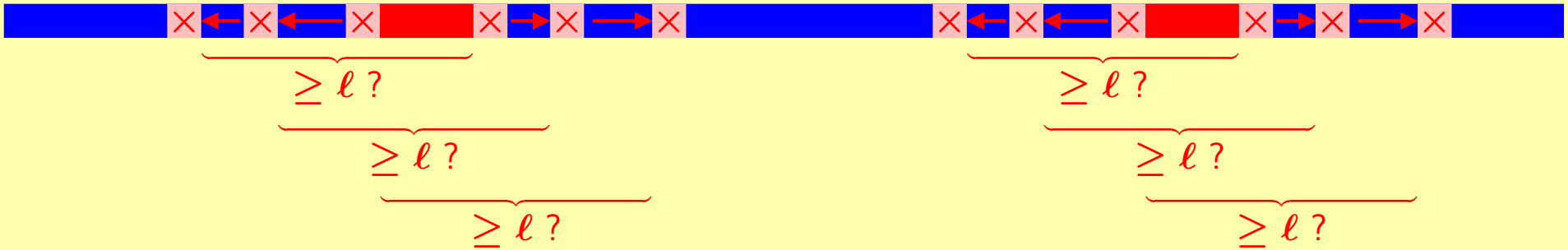
Extension for maximal k-mismatch repeats

Simple extension and length test:



Extension for maximal k-mismatch repeats

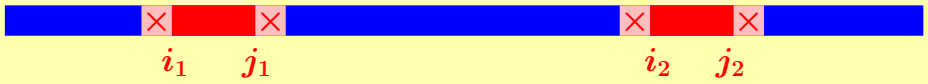
Simple extension and length test:



Analysis: $\mathcal{O}(n + \zeta k)$ time with $E(\zeta) = O\left(n^2/4^s\right)$, s minimal seed length.

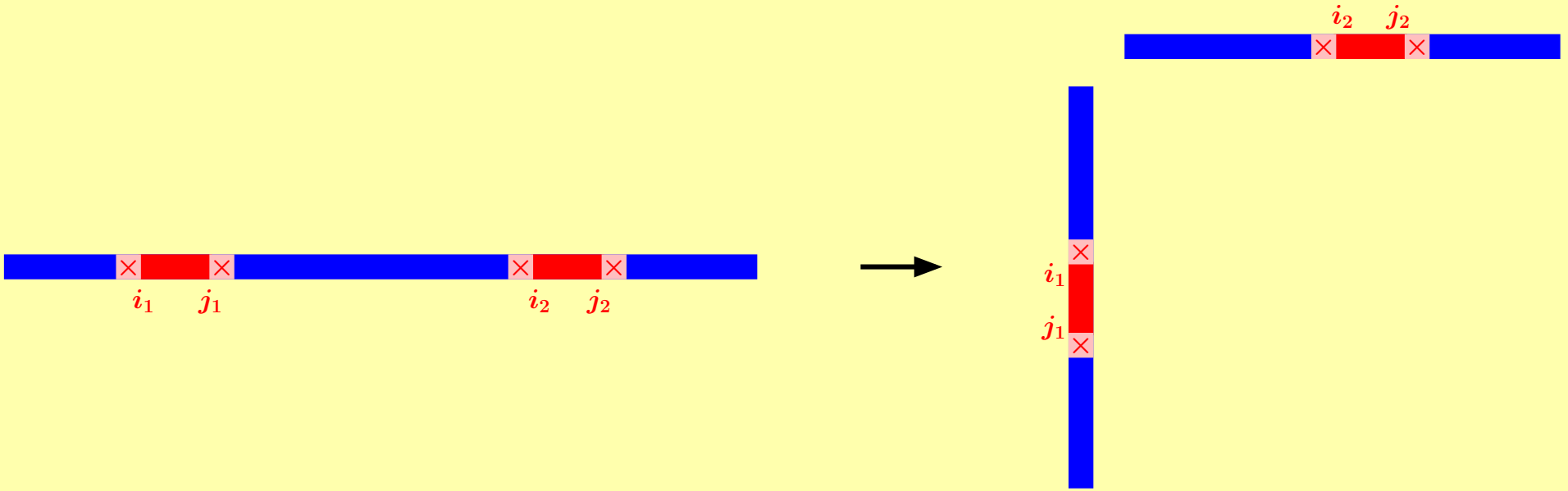
Extension for maximal k-differences repeats

Banded sequence alignment by dynamic programming:



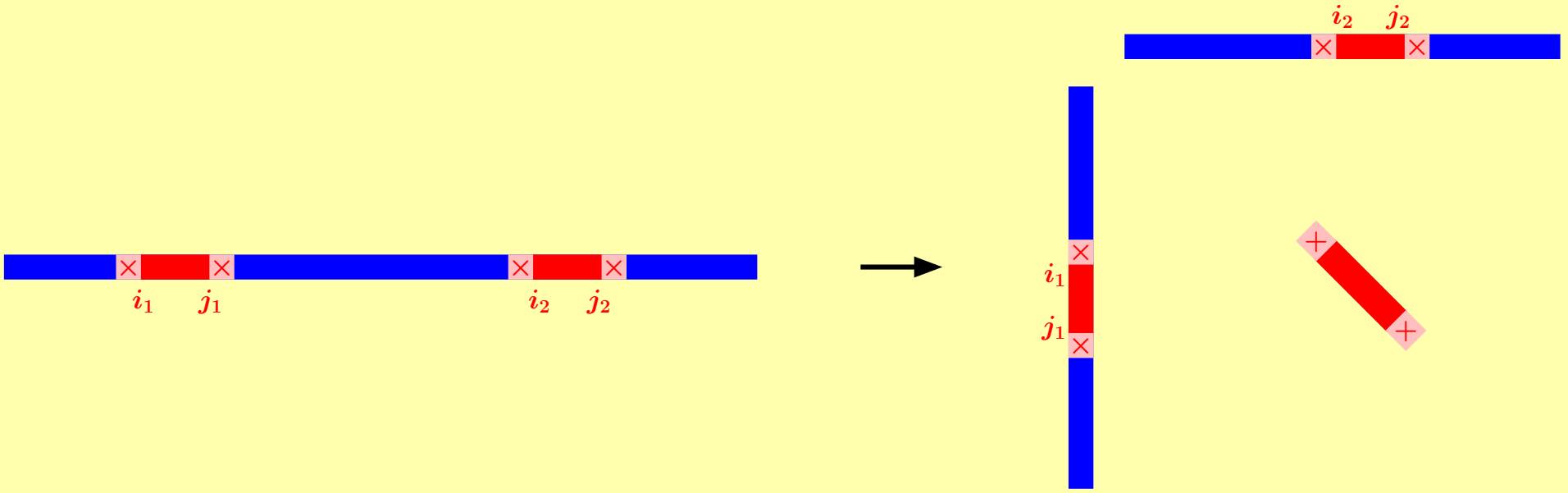
Extension for maximal k-differences repeats

Banded sequence alignment by dynamic programming:



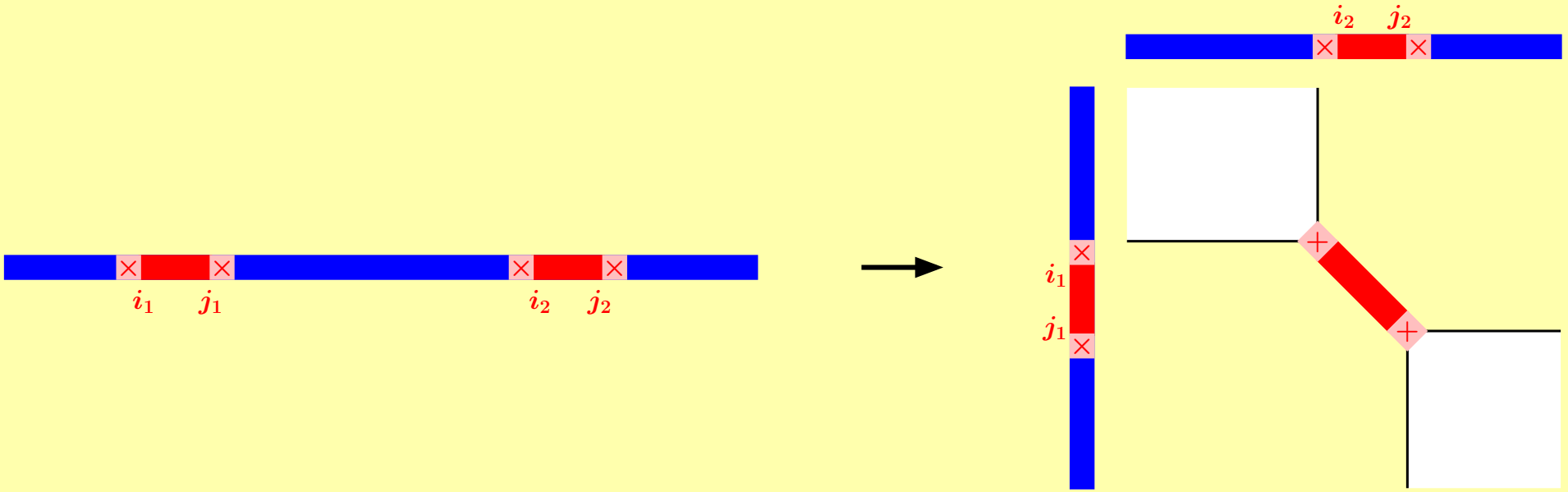
Extension for maximal k-differences repeats

Banded sequence alignment by dynamic programming:



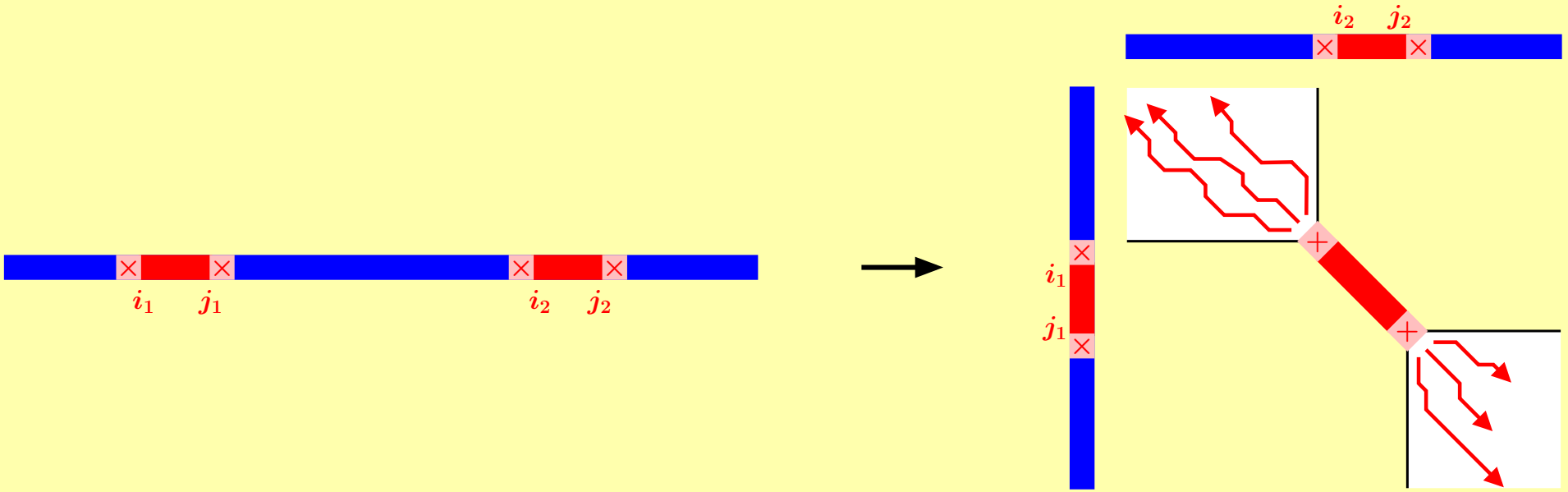
Extension for maximal k-differences repeats

Banded sequence alignment by dynamic programming:



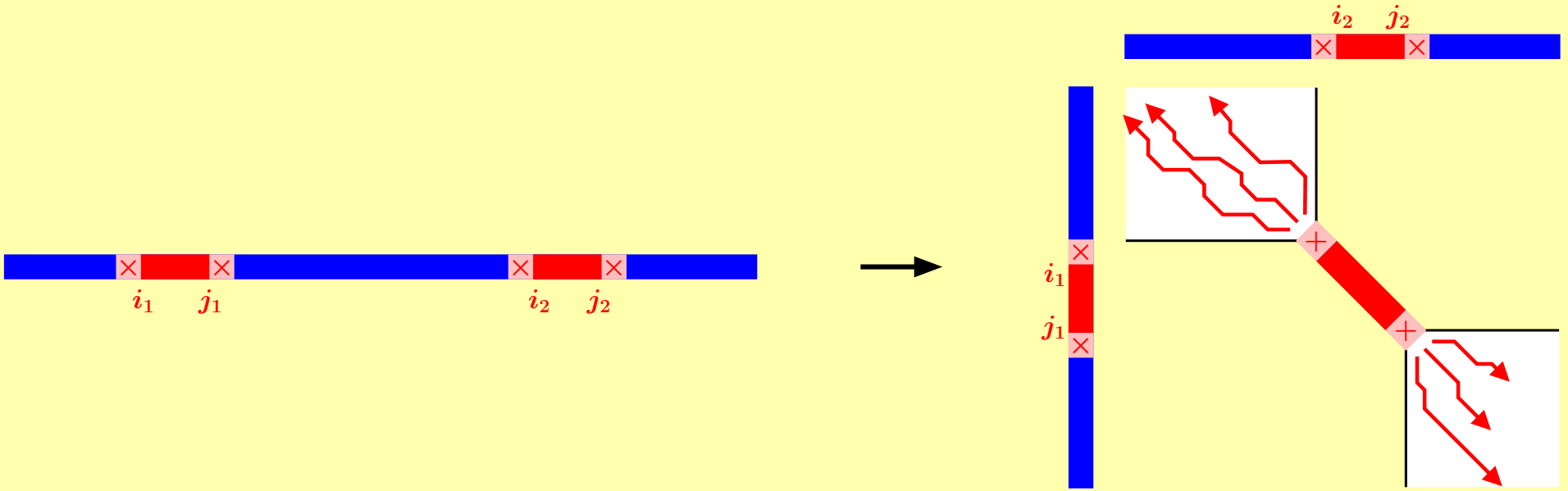
Extension for maximal k-differences repeats

Banded sequence alignment by dynamic programming:



Extension for maximal k-differences repeats

Banded sequence alignment by dynamic programming:



Analysis: $\mathcal{O}(n + \zeta k^3)$ time with $E(\zeta) = O\left(n^2/4^s\right)$, s minimal seed length.

Overview: Repeat analysis on a genomic scale

- Introduction
  - String pattern matching
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats
  - Degenerate repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

Overview: Repeat analysis on a genomic scale

- Introduction
  - String pattern matching
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats
  - Degenerate repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

The *REP*uter suite of repeat finding programs

(joint work with S. Kurtz, E. Ohlebusch, R. Giegerich, C. Schleiermacher, J. Choudhuri)

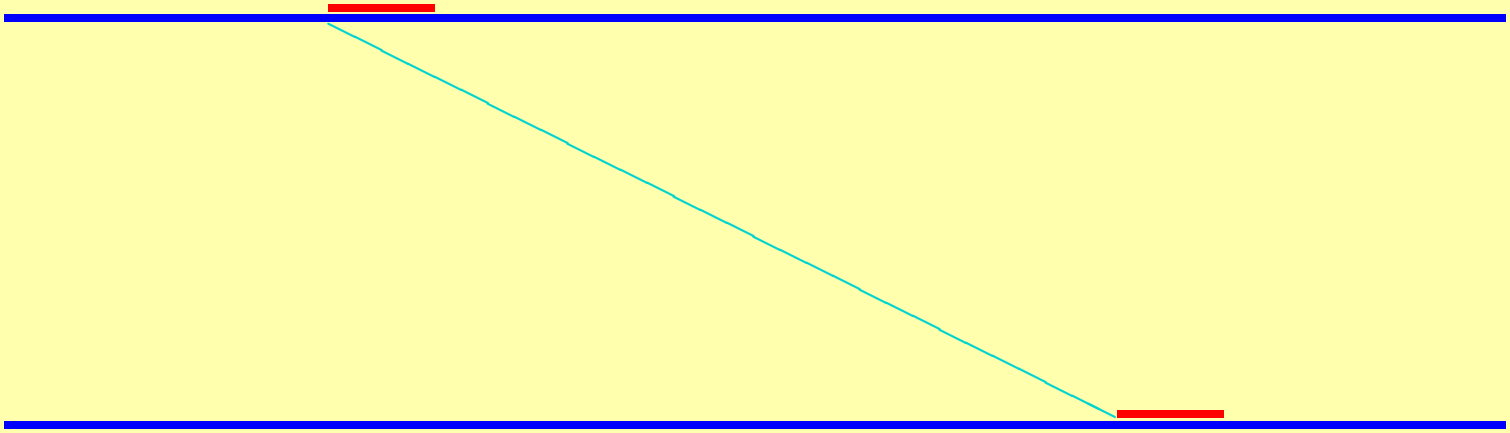
`www.genomes.de`

- *REP*find: implements several of the described algorithms.
- *REP*select: selects interesting repeats from the output of *REP*find (user-defined second filter phase).
- *REP*vis: interactive visualization tool to display large amounts of repeat data.

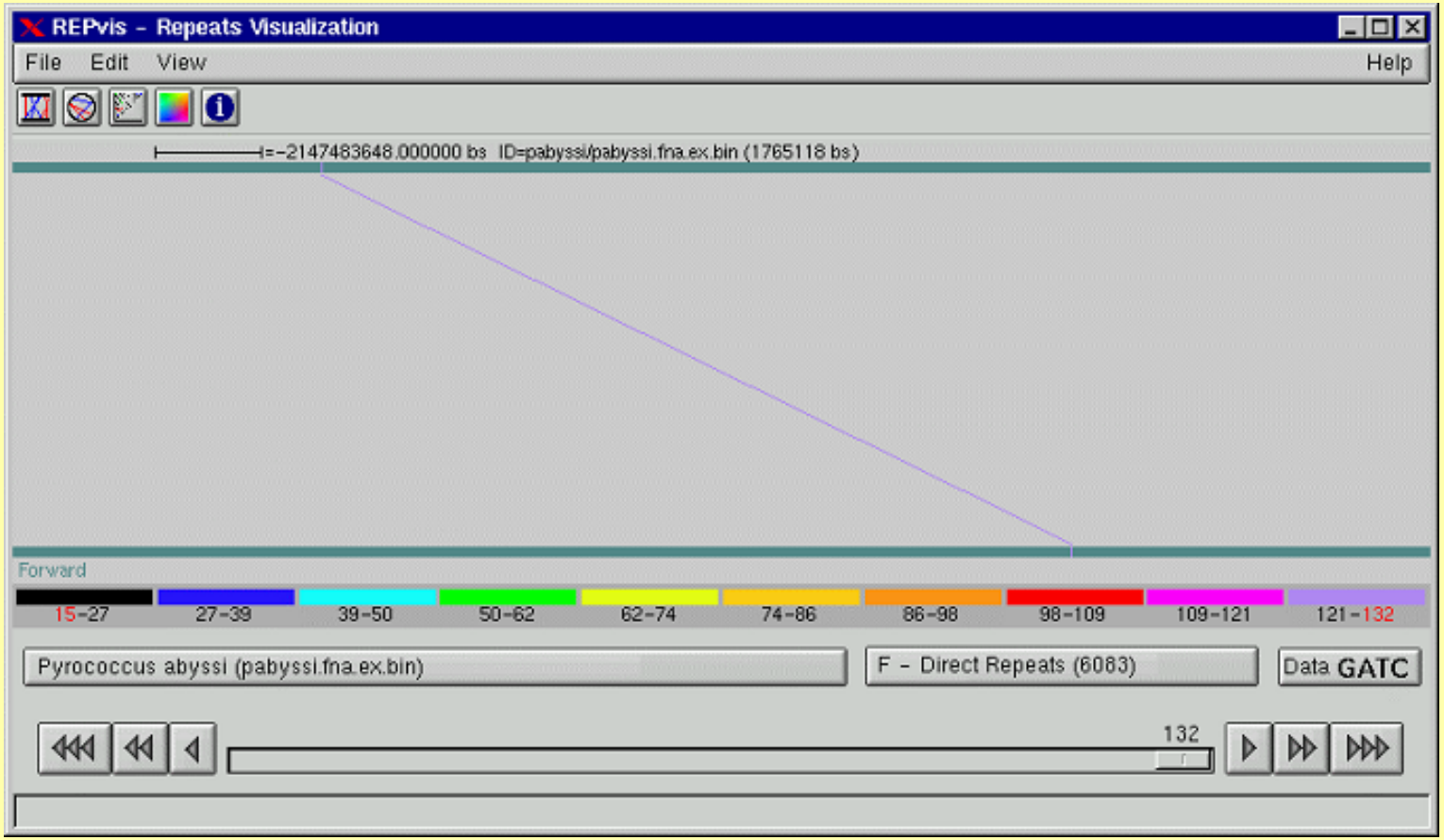
*REP*uter: An example

REPuter: An example

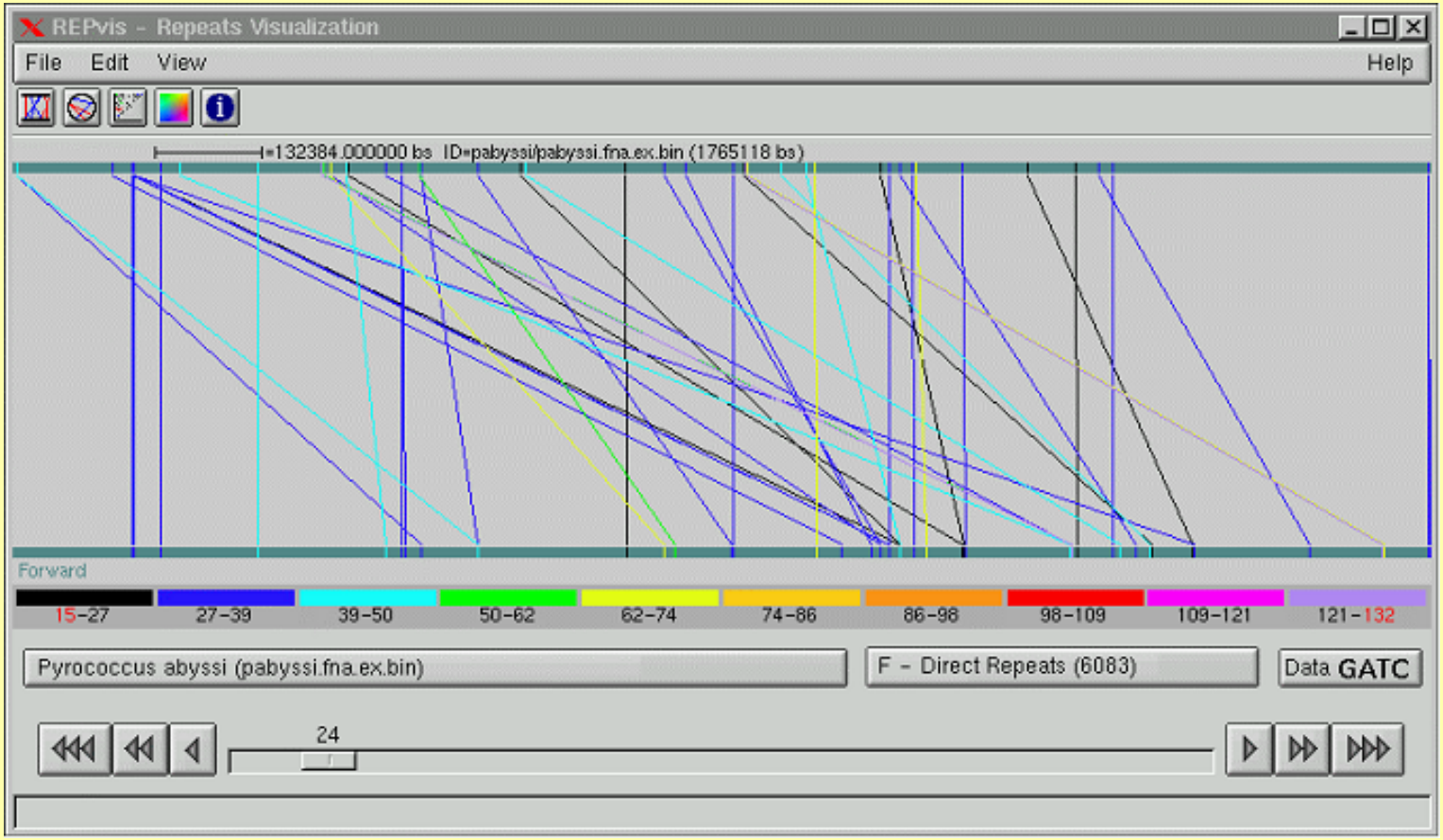
REPuter: An example



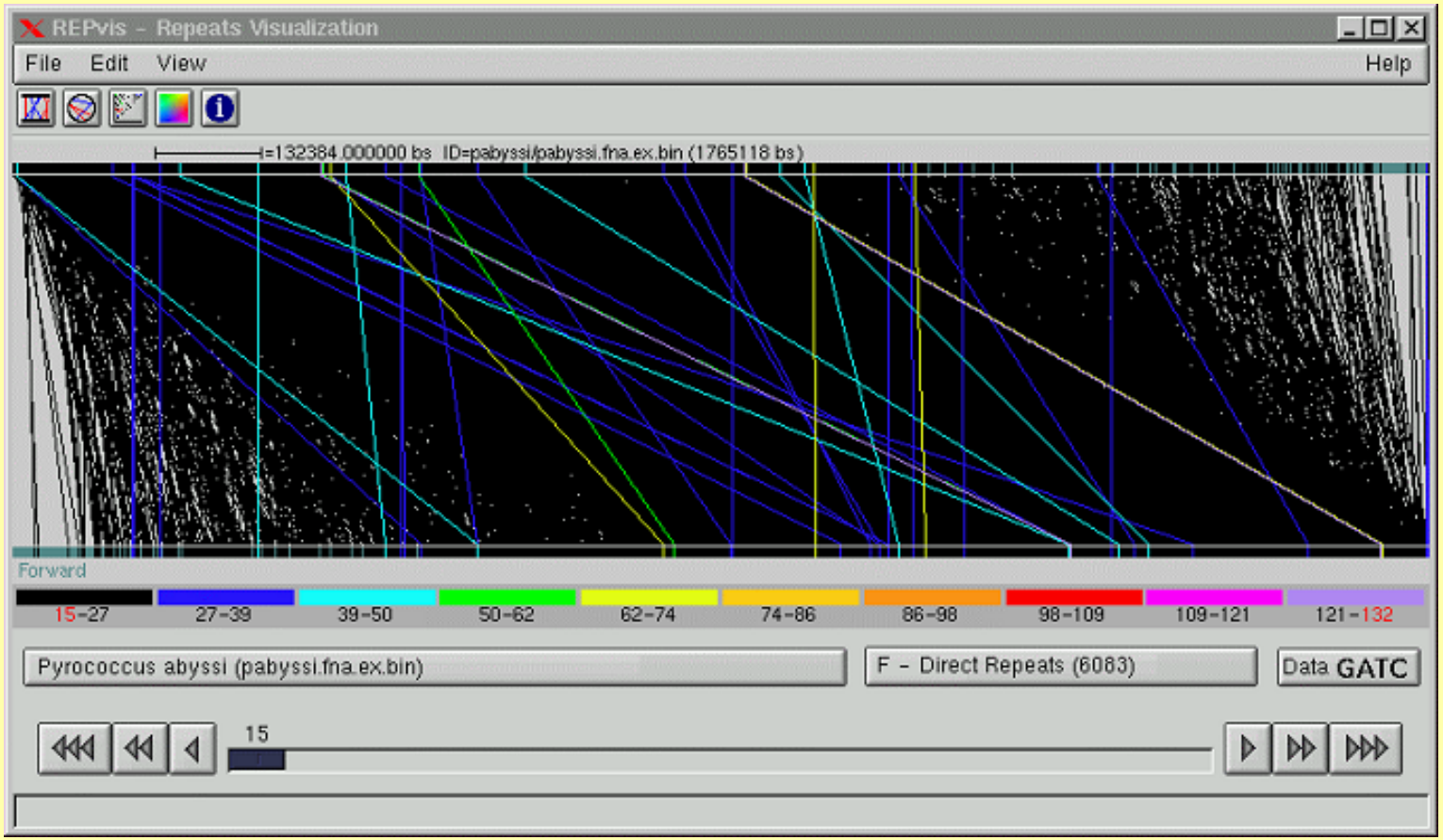
*REP*uter: An example



*REP*uter: An example

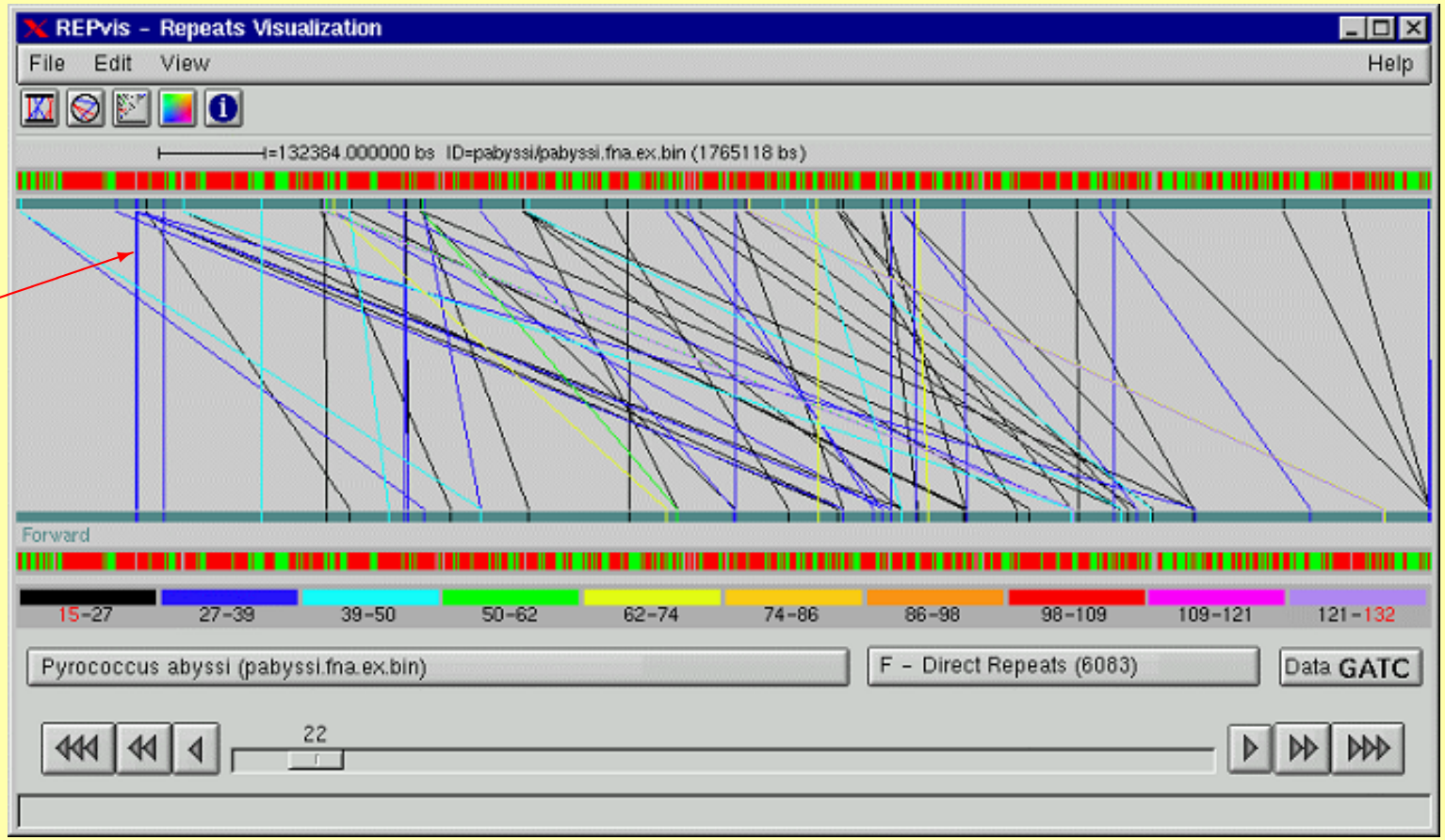


*REP*uter: An example

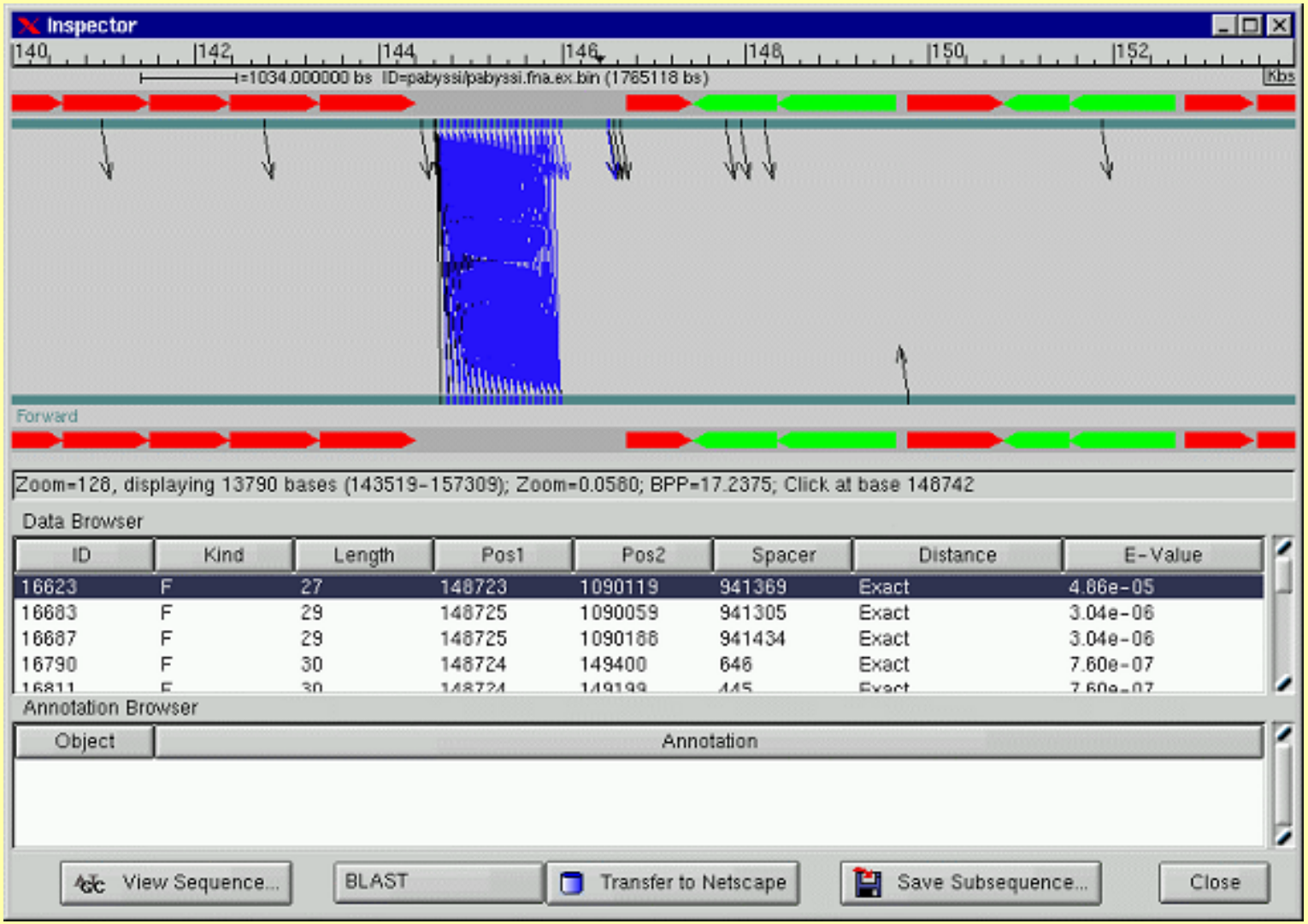


*REP*uter – Application 1: (Approximate) tandem array

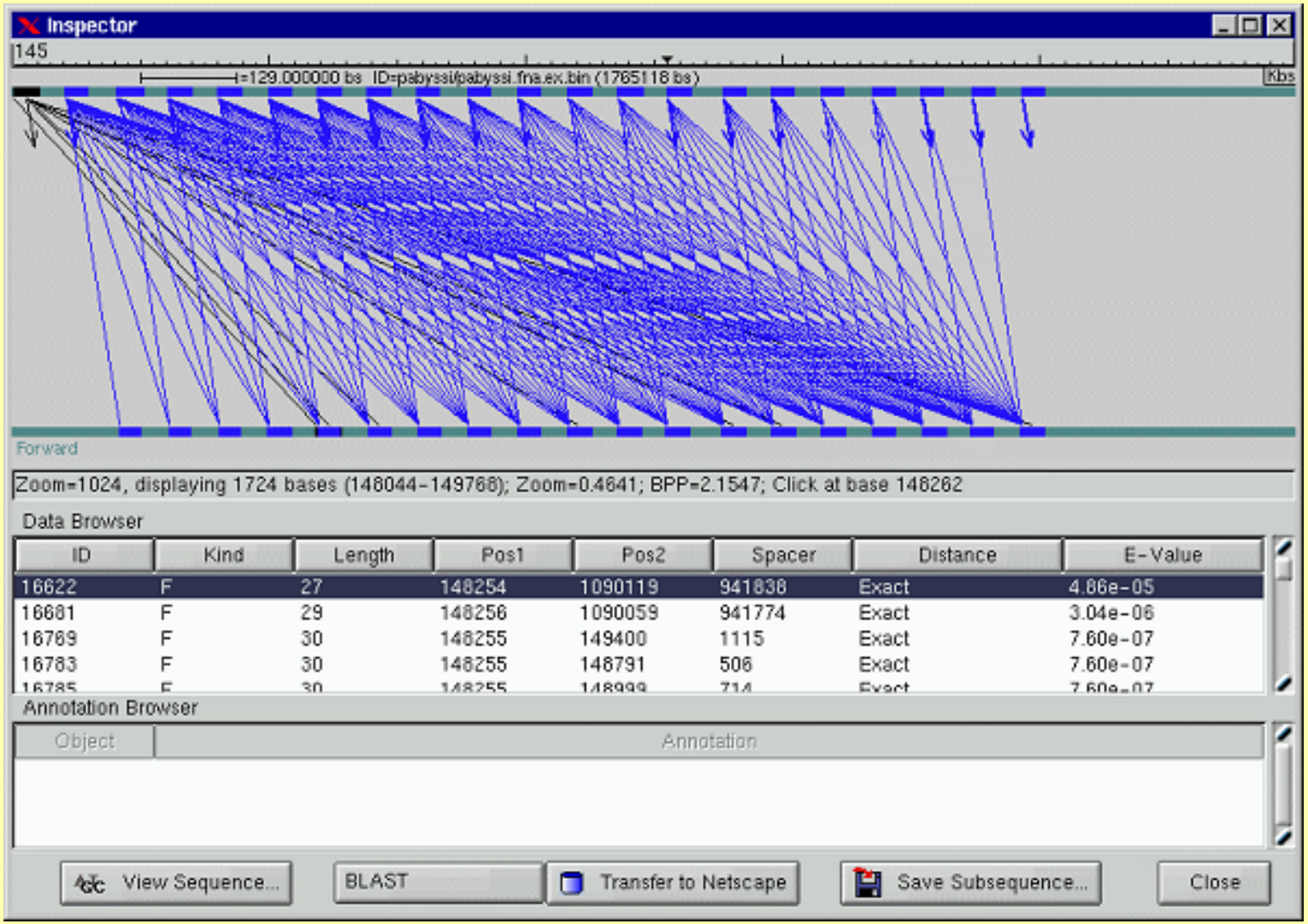
*REP*uter – Application 1: (Approximate) tandem array



*REP*uter – Application 1: (Approximate) tandem array

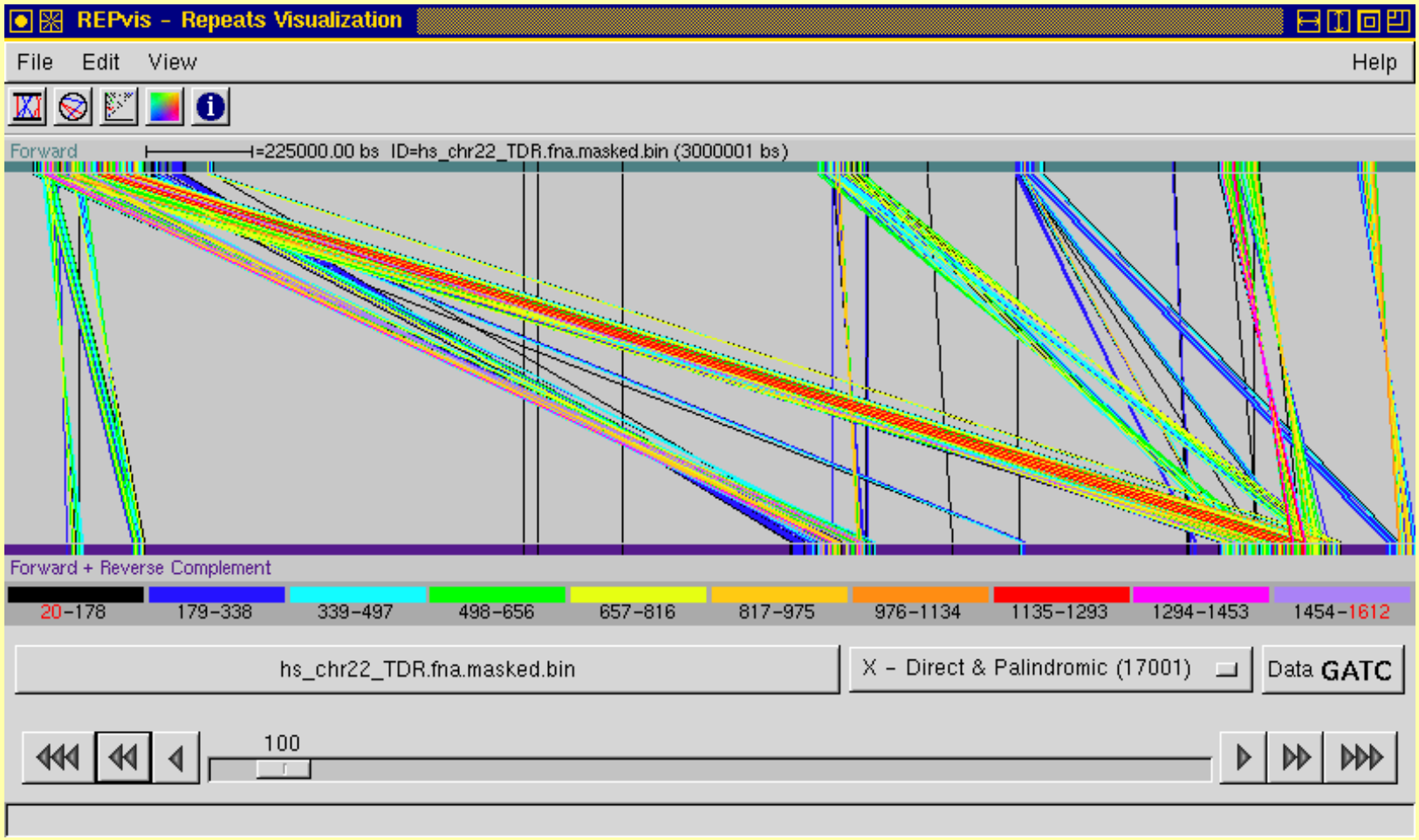


*REP*uter – Application 1: (Approximate) tandem array



REPuter – Application 2: Low copy repeats

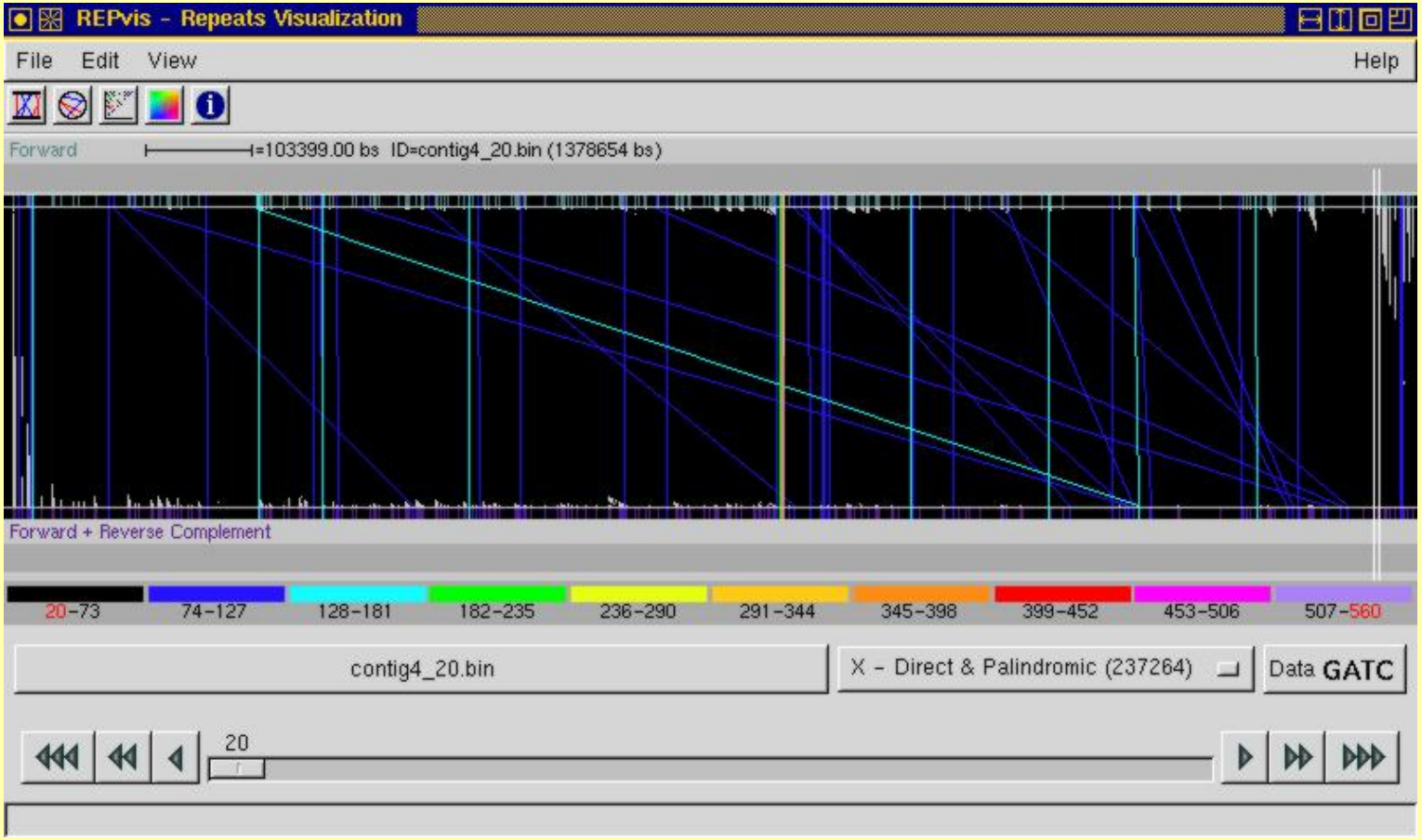
*REP*uter – Application 2: Low copy repeats



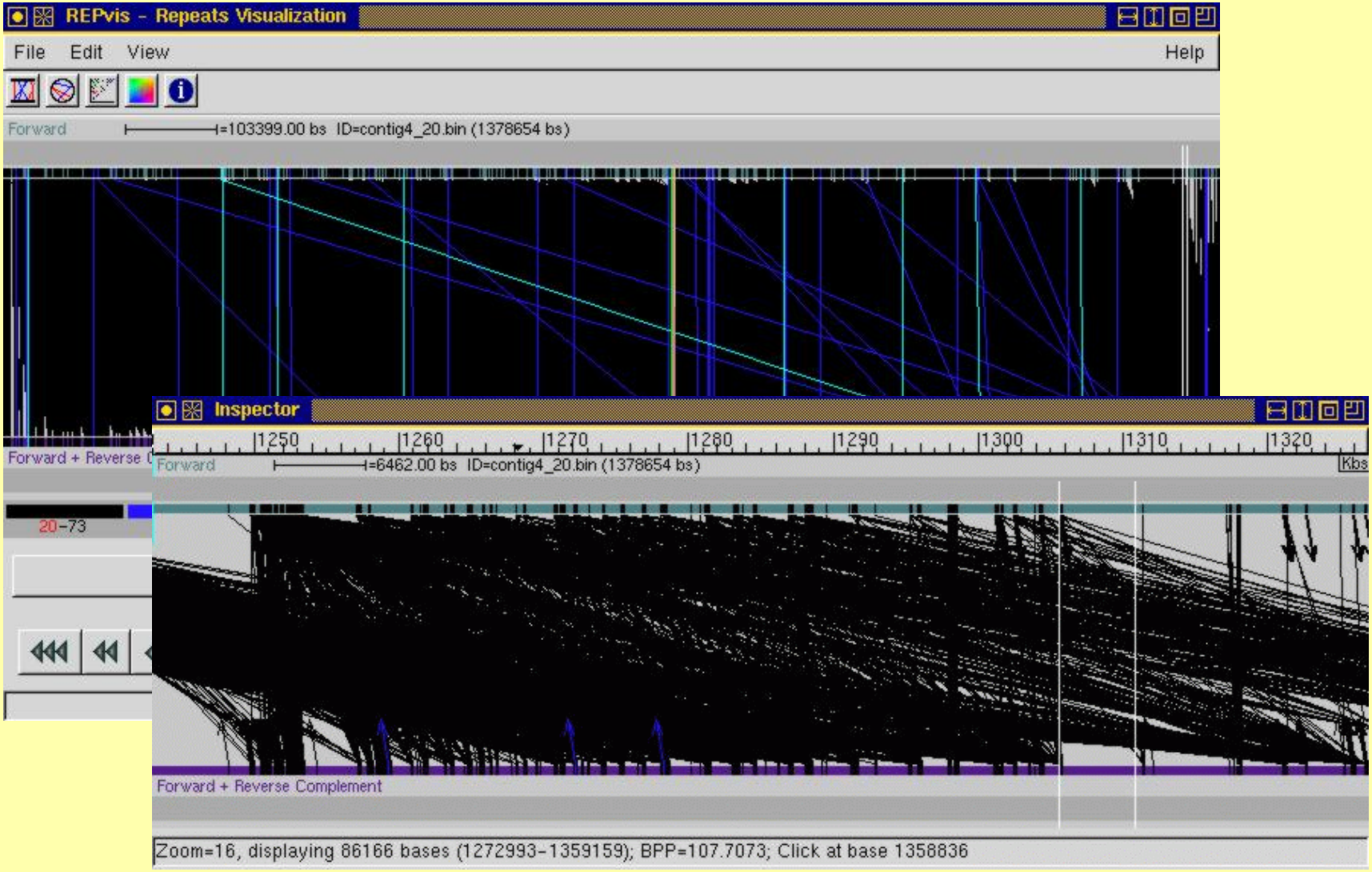
(22q11.2 region of human chromosome 22, associated to DiGeorge/Velo-cardio-facial syndrome.)

REPuter – Application 3: Unique sequences

*REP*uter – Application 3: Unique sequences



*REP*uter – Application 3: Unique sequences



*REP*uter: Computation times

genome	size [Mbases]	l [bases]	suffix tree [sec]	virtual tree [sec]
E. coli	4.42	150	5.4	1.7
S. cerevisiae	11.50	180	14.8	4.7
D. melanogaster	114.44	700	310.7	44.4

virtual (suffix) tree = suffix array, enhanced by functions to simulate suffix tree functionality $\longrightarrow$ GENalyzer.

Overview: Repeat analysis on a genomic scale

- Introduction
  - String pattern matching
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats
  - Degenerate repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

Multiple Genome Aligner

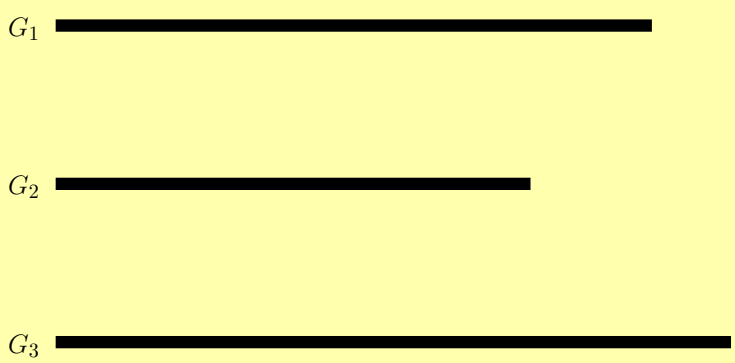
Simultaneous comparison of multiple genomes (Höhl *et al.*, ISMB 2002).

Multiple Genome Aligner

Simultaneous comparison of multiple genomes (Höhl *et al.*, ISMB 2002).

Algorithm:

1. Find all maximal multiple exact matches (multiMEMs) in the given genomes (similar to repeats, using the *generalized* suffix tree).

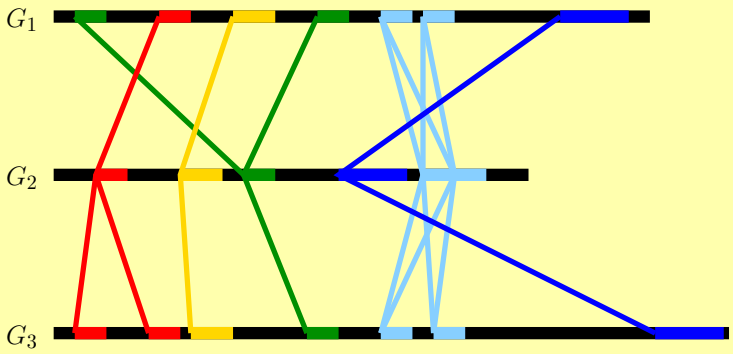


Multiple Genome Aligner

Simultaneous comparison of multiple genomes (Höhl *et al.*, ISMB 2002).

Algorithm:

1. Find all maximal multiple exact matches (multiMEMs) in the given genomes (similar to repeats, using the *generalized* suffix tree).

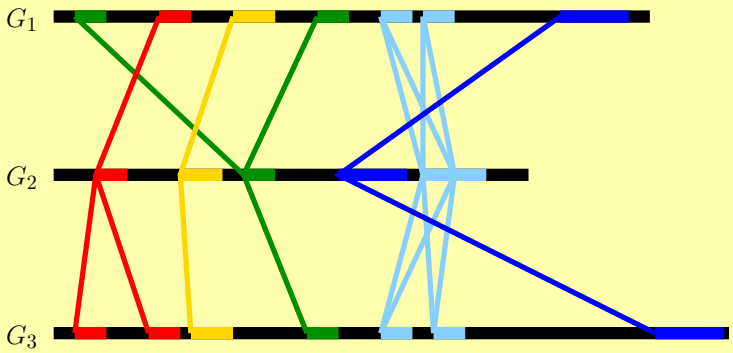


Multiple Genome Aligner

Simultaneous comparison of multiple genomes (Höhl *et al.*, ISMB 2002).

Algorithm:

1. Find all maximal multiple exact matches (multiMEMs) in the given genomes (similar to repeats, using the *generalized* suffix tree).
2. Select from all multiMEMs an optimal set,
   i.e. a chain of non-overlapping multiMEMs of maximal weight
   where the weight of a chain is the sum of the lengths of its members.

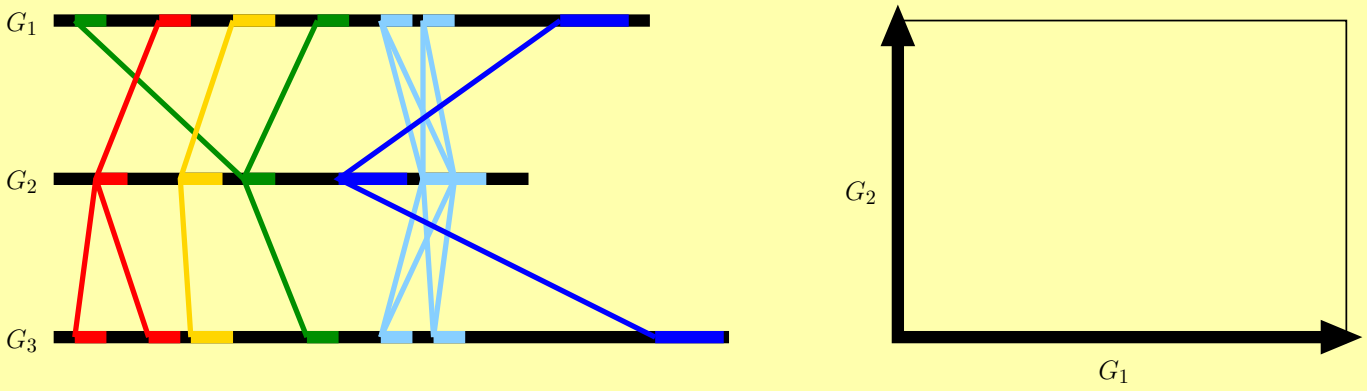


Multiple Genome Aligner

Simultaneous comparison of multiple genomes (Höhl *et al.*, ISMB 2002).

Algorithm:

1. Find all maximal multiple exact matches (multiMEMs) in the given genomes (similar to repeats, using the *generalized* suffix tree).
2. Select from all multiMEMs an optimal set,
   i.e. a chain of non-overlapping multiMEMs of maximal weight
   where the weight of a chain is the sum of the lengths of its members.

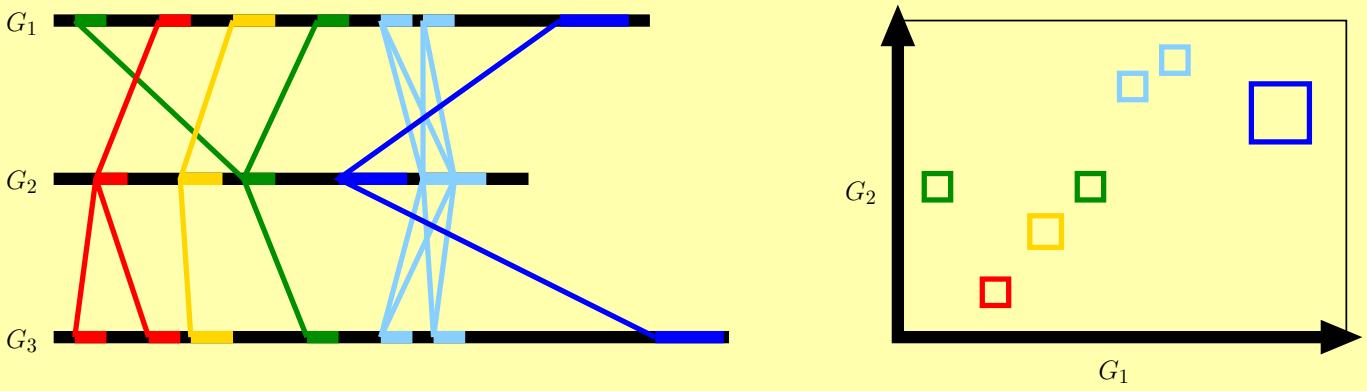


Multiple Genome Aligner

Simultaneous comparison of multiple genomes (Höhl *et al.*, ISMB 2002).

Algorithm:

1. Find all maximal multiple exact matches (multiMEMs) in the given genomes (similar to repeats, using the *generalized* suffix tree).
2. Select from all multiMEMs an optimal set,
   i.e. a chain of non-overlapping multiMEMs of maximal weight
   where the weight of a chain is the sum of the lengths of its members.

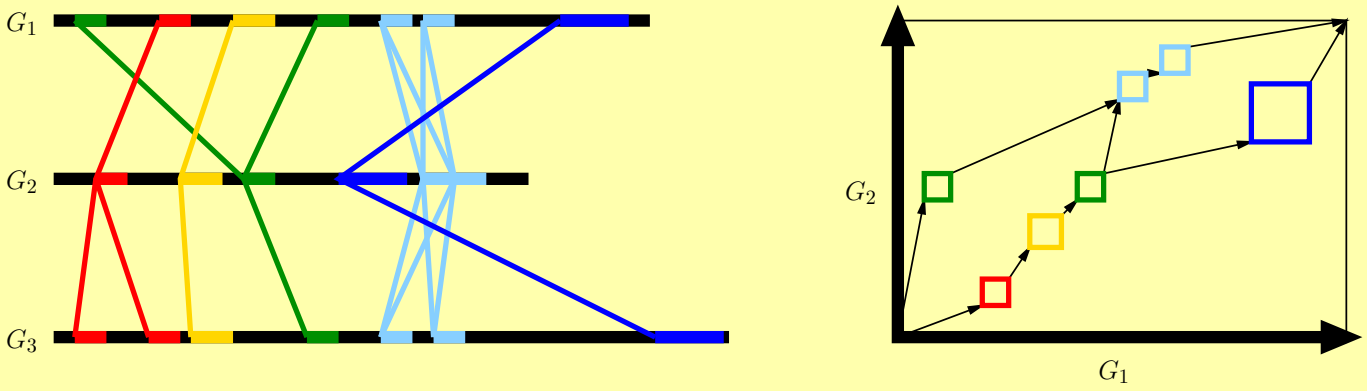


Multiple Genome Aligner

Simultaneous comparison of multiple genomes (Höhl *et al.*, ISMB 2002).

Algorithm:

1. Find all maximal multiple exact matches (multiMEMs) in the given genomes (similar to repeats, using the *generalized* suffix tree).
2. Select from all multiMEMs an optimal set,
   i.e. a chain of non-overlapping multiMEMs of maximal weight
   where the weight of a chain is the sum of the lengths of its members.

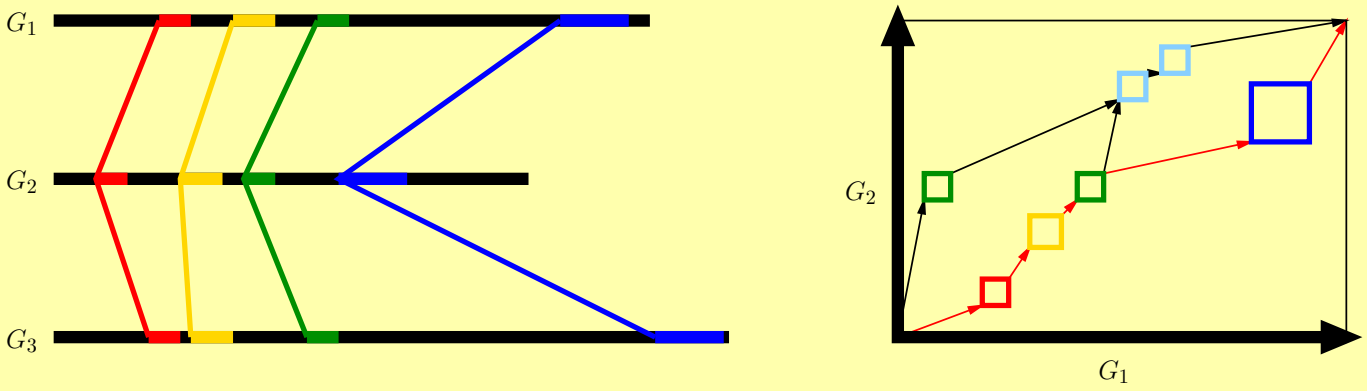


Multiple Genome Aligner

Simultaneous comparison of multiple genomes (Höhl *et al.*, ISMB 2002).

Algorithm:

1. Find all maximal multiple exact matches (multiMEMs) in the given genomes (similar to repeats, using the *generalized* suffix tree).
2. Select from all multiMEMs an optimal set,
   i.e. a chain of non-overlapping multiMEMs of maximal weight
   where the weight of a chain is the sum of the lengths of its members.

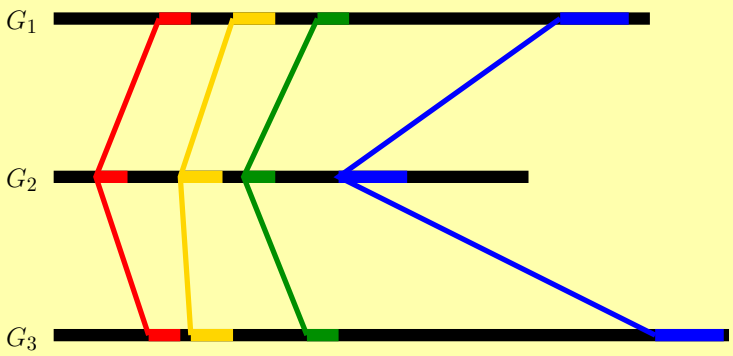


Multiple Genome Aligner

Simultaneous comparison of multiple genomes (Höhl *et al.*, ISMB 2002).

Algorithm:

1. Find all maximal multiple exact matches (multiMEMs) in the given genomes (similar to repeats, using the *generalized* suffix tree).
2. Select from all multiMEMs an optimal set,
   i.e. a chain of non-overlapping multiMEMs of maximal weight
   where the weight of a chain is the sum of the lengths of its members.

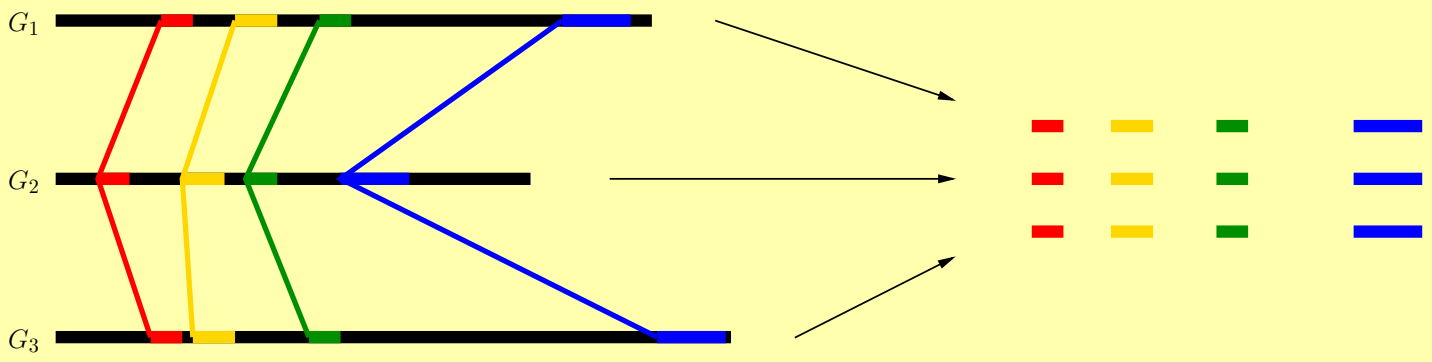


Multiple Genome Aligner

Simultaneous comparison of multiple genomes (Höhl *et al.*, ISMB 2002).

Algorithm:

1. Find all maximal multiple exact matches (multiMEMs) in the given genomes (similar to repeats, using the *generalized* suffix tree).
2. Select from all multiMEMs an optimal set,
   i.e. a chain of non-overlapping multiMEMs of maximal weight
   where the weight of a chain is the sum of the lengths of its members.
3. Close the gaps recursively, and finally by a standard alignment procedure.

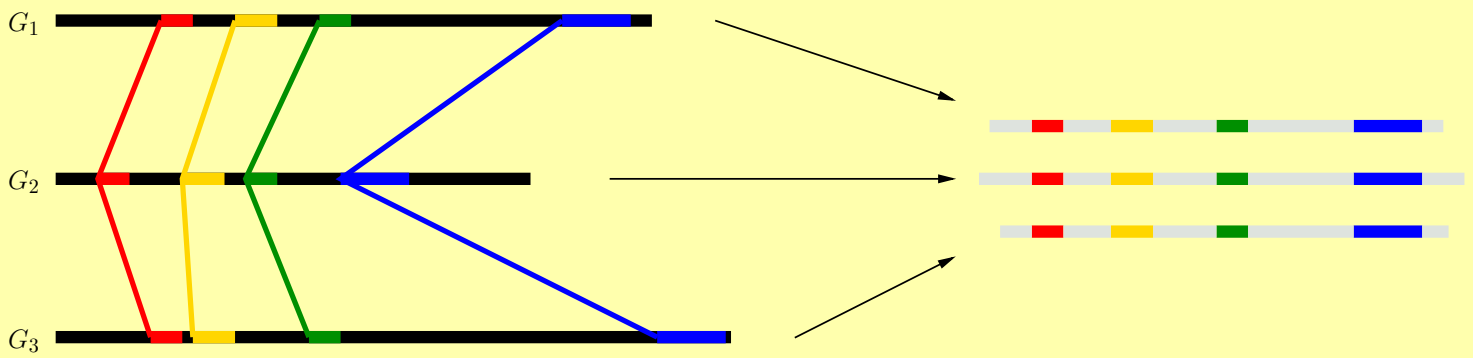


Multiple Genome Aligner

Simultaneous comparison of multiple genomes (Höhl *et al.*, ISMB 2002).

Algorithm:

1. Find all maximal multiple exact matches (multiMEMs) in the given genomes (similar to repeats, using the *generalized* suffix tree).
2. Select from all multiMEMs an optimal set,
   i.e. a chain of non-overlapping multiMEMs of maximal weight
   where the weight of a chain is the sum of the lengths of its members.
3. Close the gaps recursively, and finally by a standard alignment procedure.



Overview: Repeat analysis on a genomic scale

- Introduction
  - String pattern matching
  - Suffix trees
  - Repeats
- Repeat finding with suffix trees
  - Exact repeats
  - Tandem repeats
  - Degenerate repeats
- Bioinformatics tools and applications
  - REPuter
  - Multiple genome aligner (MGA)
- Conclusion

Summary: Repeats and suffix trees

Some results: (z is always the output size)

- Find all z maximal repeats in $\mathcal{O}(n+z)$ time.
- Find all z maximal palindromic repeats in $\mathcal{O}(n+z)$ time.
- Find all z tandem repeats in $\mathcal{O}(n \log n + z)$ time or $\mathcal{O}(n+z)$ time.
- Find all z maximal repeats with bounded gap in $\mathcal{O}(n \log n + z)$ time.
- Find all z maximal repeats with lower-bounded gap in $\mathcal{O}(n+z)$ time.
- Find all degenerate repeats with $\leq k$ errors in $\mathcal{O}(n + \zeta k^3)$ time ($E(\zeta) = \mathcal{O}\left(n^2/4^s\right)$).

Conclusion

- Repeat finding on a whole genome/whole chromosome basis is possible.
- Suffix trees are a powerful data structure (not only) for repeat finding.
- The flexibility of the data structures allows to find various (related) types of repeats.
- Repeat finding has several applications, some of which are related to repeats only at second glance.

Conclusion

The analysis of biological sequence data produces several interesting computational questions. Several CS disciplines involved:

- Algorithms and data structures
- Algorithm engineering
- Software engineering
- Visualization

Not only does Biology profit from Computer Science, but also vice versa!

Acknowledgments

Suffix trees

- Stefan Kurtz (Hamburg)
- Robert Giegerich (Bielefeld)

Repeats

- Dan Gusfield (Davis)
- Christian N. S. Pedersen, Gerth S. Brodal (Århus)
- Rune B. Lyngsø (Oxford)
- Enno Ohlebusch (Ulm)
- Chris Schleiermacher (Artemis Pharmaceuticals)
- Jomuna Choudhuri (Bielefeld)

Black slide