Kap. 3: Hierarchische Zeichenverfahren
3.3 Zählen von Kreuzungen

Prof. Dr. Petra Mutzel

Lehrstuhl für
Algorithm Engineering LS11
Universität Dortmund

6. VO WS07/08 30. Oktober 2007

Literatur für diese VO

- W. Barth, M. Jünger und P. Mutzel: Simple and Efficient 2-Layer Cross Counting, Journal of Graph Algorithms and Applications (JGAA), 2003

- H. Nagamochi und N. Yamada: Counting Edge Crossings in a 2-Layered Drawing. Information Processing Letters 91, 221-225, 2004

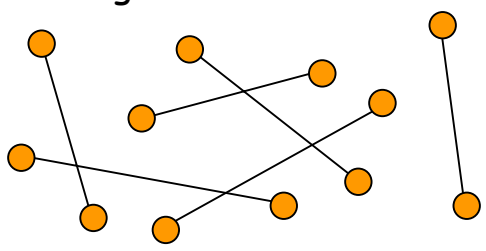
Überblick zu Kapitel 3

3.1 Einführung und Überblick ✓
3.2 Schichtzuweisung ✓
3.3 Zählen von Kreuzungen
3.4 Kreuzungsminimierung
3.5 Koordinatenzuweisung

Überblick: Zählen von Kreuzungen

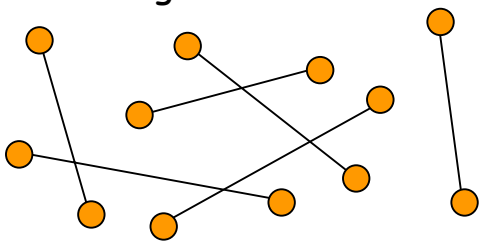
- Einführung
- Algorithmen
  - Sweepline Algorithmus
  - BJM-Algorithmus und Variationen
- Experimentelle Resultate
- Algorithmen von Nagamochi & Yamada NEU

Allgemeines Problem



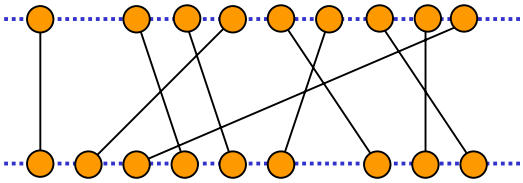
Gegeben: Menge S von Geradensegmenten
Gesucht: Anzahl der Schnittpunkte

Allgemeiner Fall



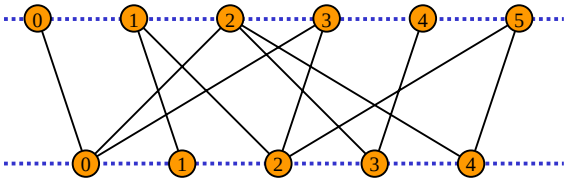
Ausgeben der Kreuzungen:
Chazelle & Edelsbrunner [1992] $O(|E| \log |E| + |C|)$
Nur Zählen:
Chazelle [1985] $O(|E|^{1.695})$

Spezielles Problem



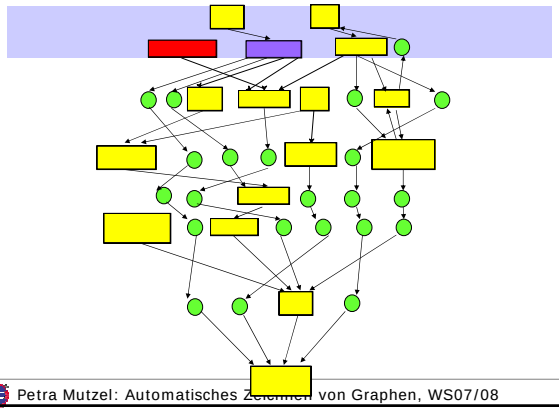
Endpunkte der Segmente
liegen auf parallelen Geraden

...eigentlich: bipartiter Graph

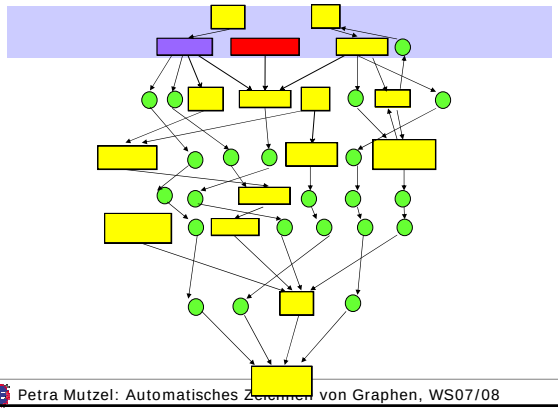


Anwendung:
Automatisiertes Zeichnen von Graphen

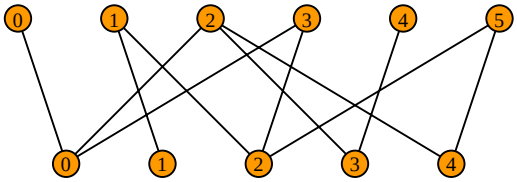
Kreuzungsreduktion



Kreuzungsreduktion

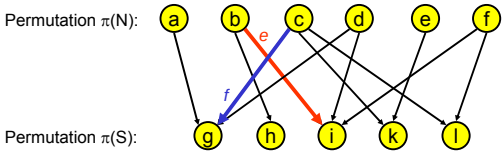


Zählen der Kreuzungen



Vor und nach jeder Heuristik-Anwendung:
Zählen von Kreuzungen

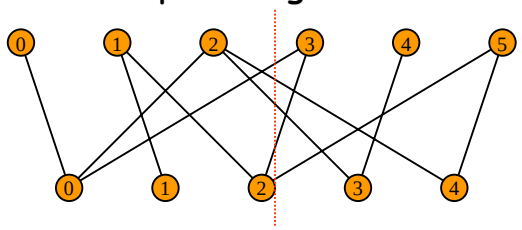
Naiver Algorithmus



Für alle Kantenpaare $e,f \in E$:
Falls $\pi(e_N) < \pi(f_N)$ UND $\pi(e_S) > \pi(f_S)$
Dann: Zähle Kreuzung

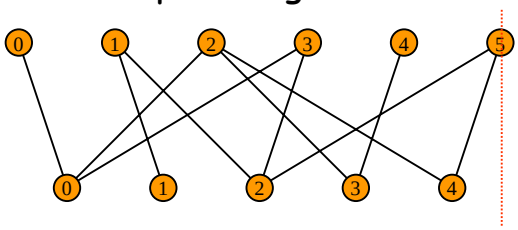
Laufzeit: $O(|E|^2)$

Sweepline Algorithmus



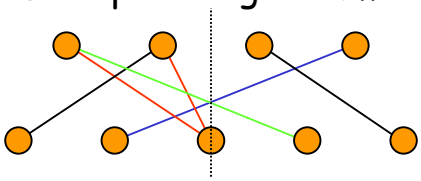
6

Sweepline Algorithmus



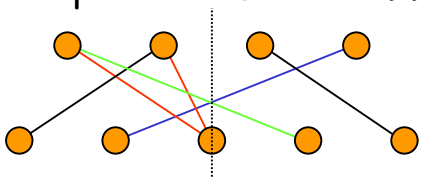
12

Sweepline Algorithmus



Bewege Sweepline von links nach rechts:
Teilung in
- bereits beendete Kanten (links),
- noch nicht erreichte Kanten (rechts), und
- aktive Kanten (1 Endknoten erreicht)

Sweepline von Sander 1995



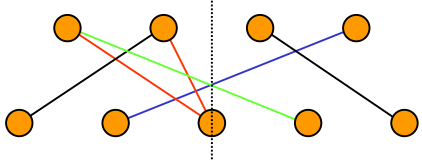
Zwei geordnete Listen UL und LL:
Endknoten der aktiven Kanten wird in der
- UL-Liste in der oberen Schicht deaktiviert
- LL-Liste in der unteren Schicht deaktiviert

Laufzeit: $O(|E| + |C|)$

Sweepline von Sander 1995

1. **Vorverarbeitung:** Splitte Knoten, so dass Knotengrad = 1 für alle Knoten.
2. Durchlaufe die Knoten nacheinander (von links nach rechts).
3. **Fall A:** Sei v der aktuelle Knoten in unteren Schicht von Kante (w,v)
4. Falls v Endknoten ist, dann
5. CrossCount += |UL|
6. u = first(LL)
7. While u links von w liegt {
8. CrossCount += 1; u = next(LL)
9. }
10. /* jetzt gilt: u==w /* entferne w aus Liste UL
11. Sonst: aktiviere neue Kante: Eintrag in Liste UL
12. **Fall B:** vertausche Rollen von UL und LL

Geht es schneller als O(|E|+|C|)?

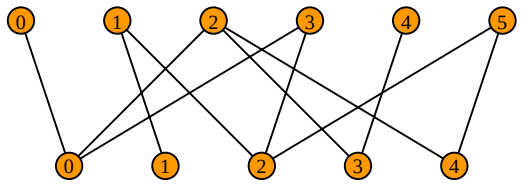


Anzahl der Kreuzungen:
- Worst Case: $|E|^2$
- Average Case: $¼ |E|^2$

Auflisten von Kreuzungen: **NEIN!**

Zählen von Kreuzungen: **JA!**

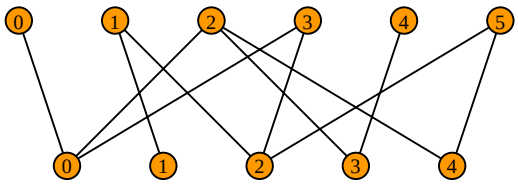
Waddle & Malhotra



Ein zweiter Sweepline Algorithmus:
$O(|E| \log |V|)$ Waddle & Malhotra [1999]

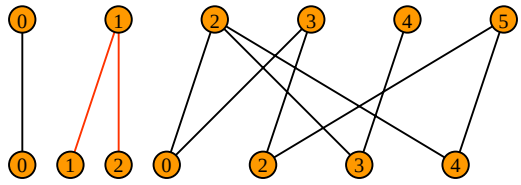
Probleme:
- viele komplizierte Fallunterscheidungen
- aufwändiger Algorithmus (viele Seiten)
- aufwändige Implementierung

Barth, Jünger & Mutzel

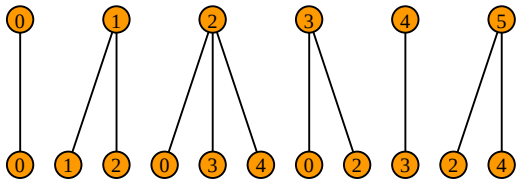


Ein einfacher
$O(|E| \log |V|)$ Algorithmus

Lexikographisches Sortieren

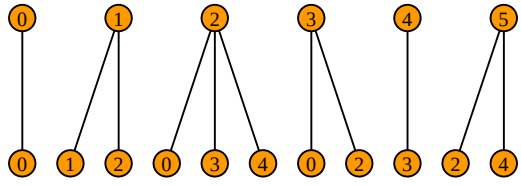


Lexikographisches Sortieren



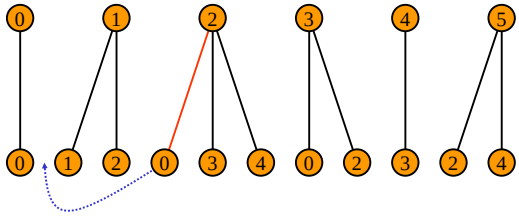
Beobachtung

Die Anzahl der Kreuzungen ist gleich der Anzahl der Inversionen der unteren Folge.



Zählen der Inversionen

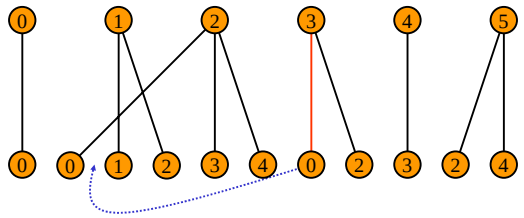
Insertion Sort



Kreuzungen: 2

Zählen der Inversionen

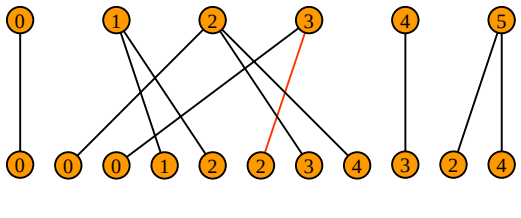
Insertion Sort



Kreuzungen: 2 + 4

Zählen der Inversionen

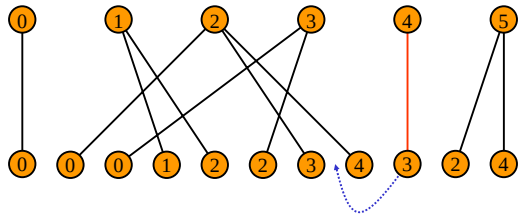
Insertion Sort



Kreuzungen: 2 + 4 + 2

Zählen der Inversionen

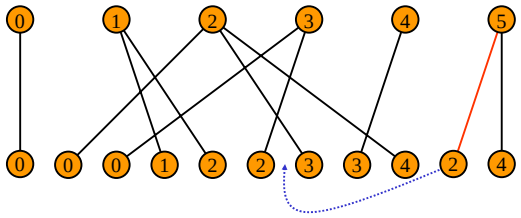
Insertion Sort



Kreuzungen: 2 + 4 + 2 + 1

Zählen der Inversionen

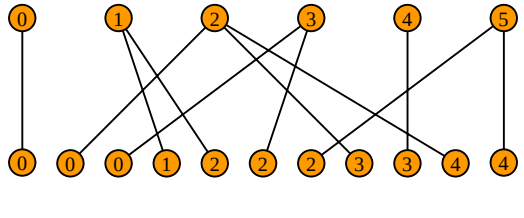
Insertion Sort



Kreuzungen: 2 + 4 + 2 + 1 + 3

Zählen der Inversionen

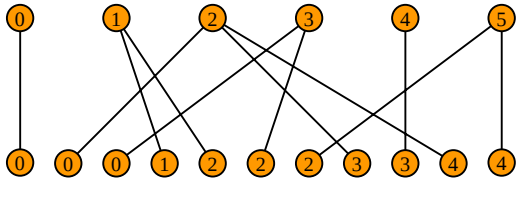
Insertion Sort



Kreuzungen: 2 + 4 + 2 + 1 + 3 = 12

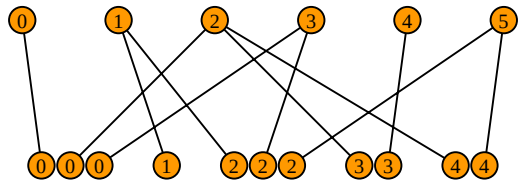
Zählen der Inversionen

Insertion Sort



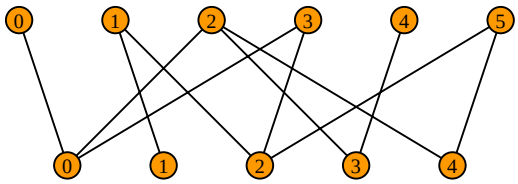
Zählen der Inversionen

Insertion Sort



Zählen der Inversionen

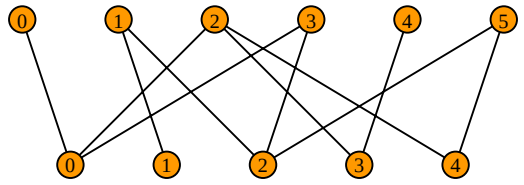
Insertion Sort



Zählen der Inversionen

Insertion Sort

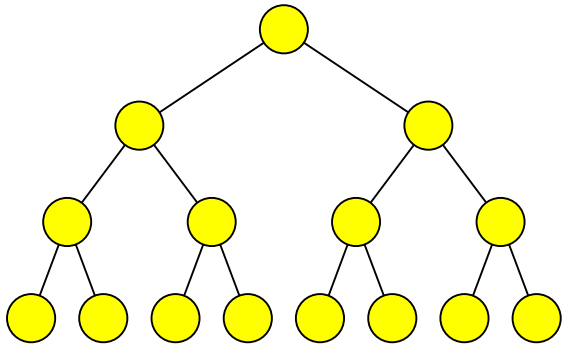
Laufzeit: O(|E|+|C|)

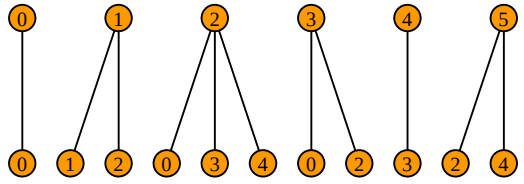


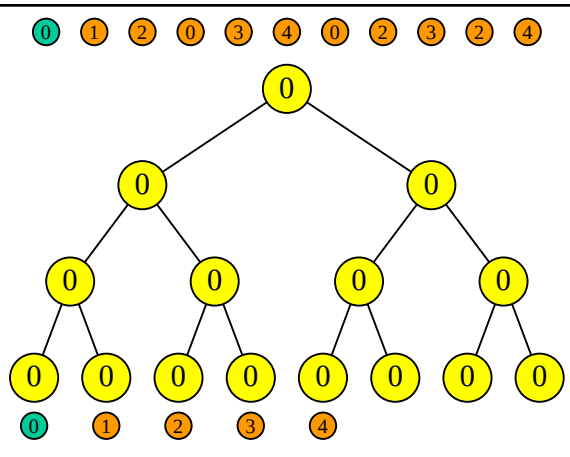
Alternative 1: Merge Sort: O(|E| log |E|)

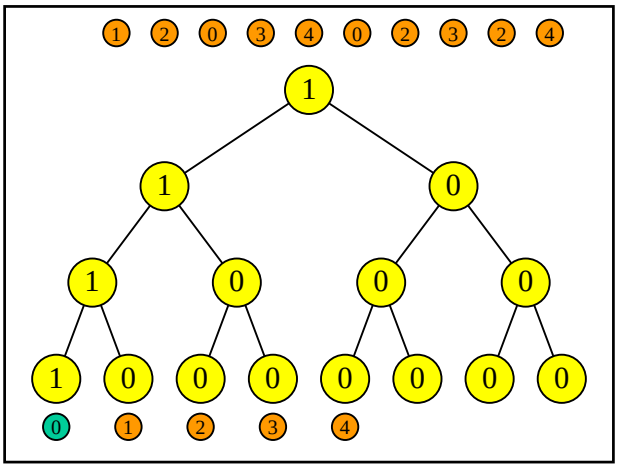
Alternative 2: Verbesserung auf O(|E| log $|V_{small}|$)

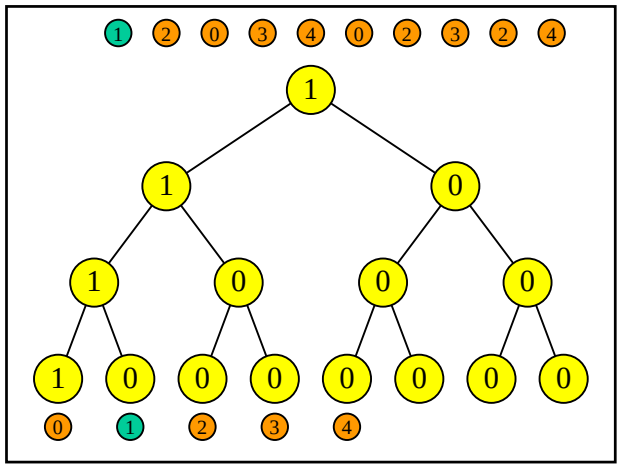
Accumulator Tree

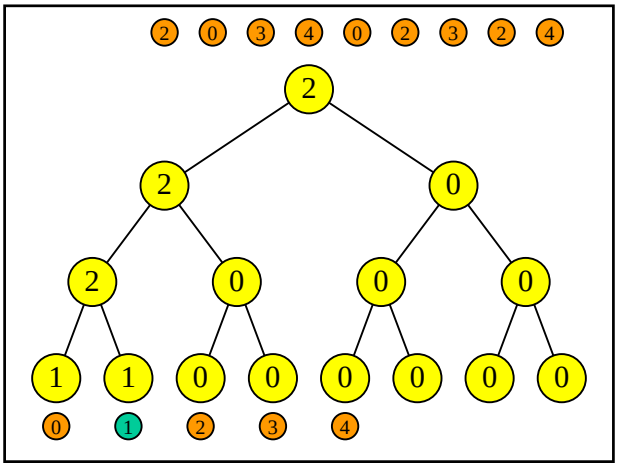


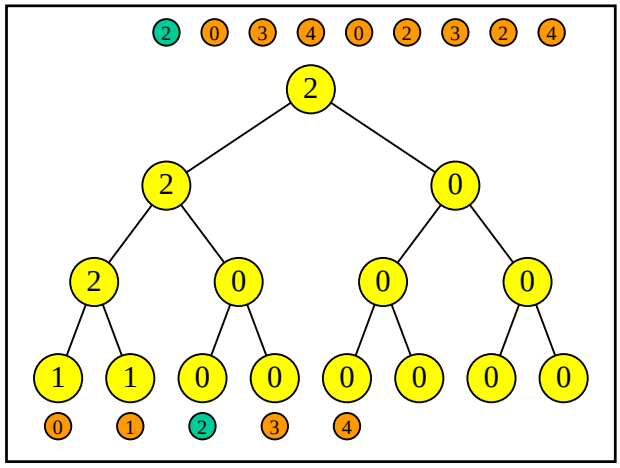


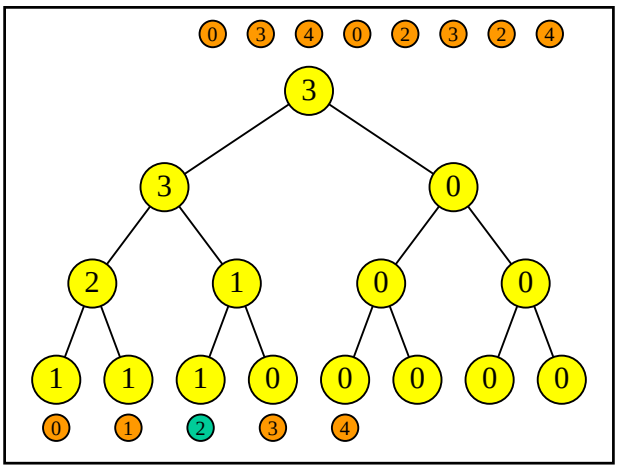


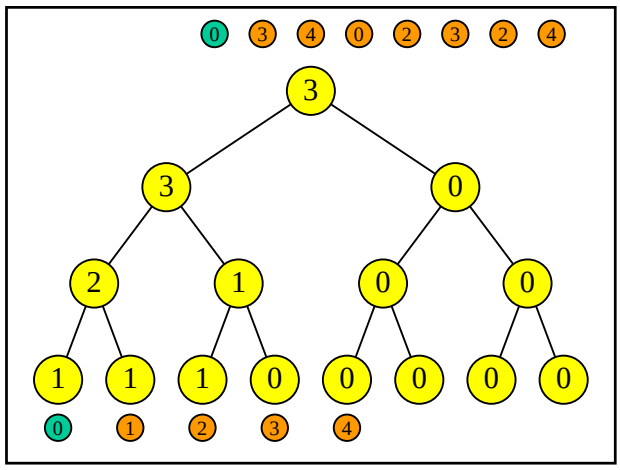

1 2 0 3 4 0 2 3 2 4
1
1 0
1 0 0 0
1 0 0 0 0 0 0 0
0 1 2 3 4

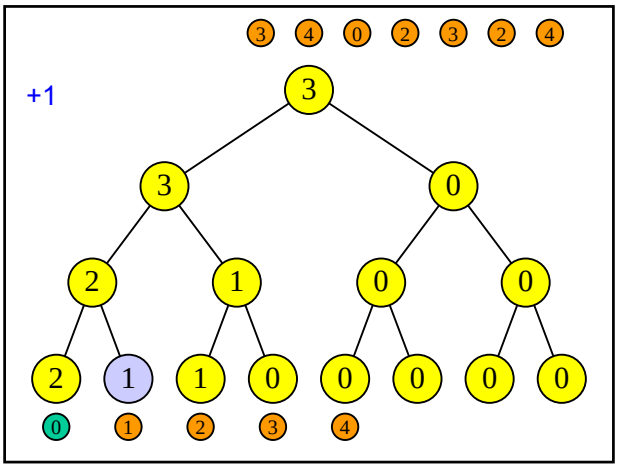

1 2 0 3 4 0 2 3 2 4
1
1 0
1 0 0 0
1 0 0 0 0 0 0 0
0 1 2 3 4

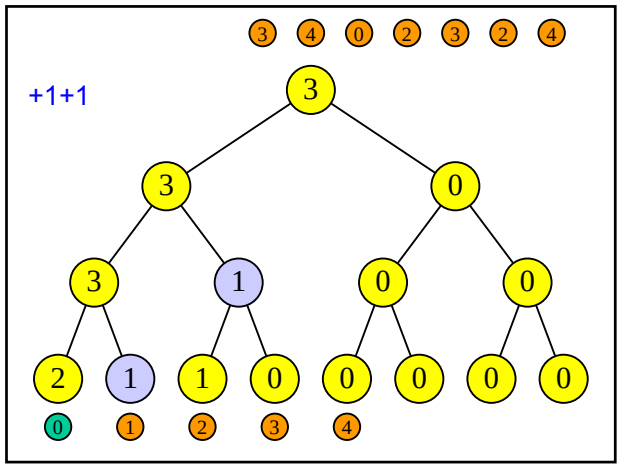

2 0 3 4 0 2 3 2 4
2
2 0
2 0 0 0
1 1 0 0 0 0 0 0
0 1 2 3 4

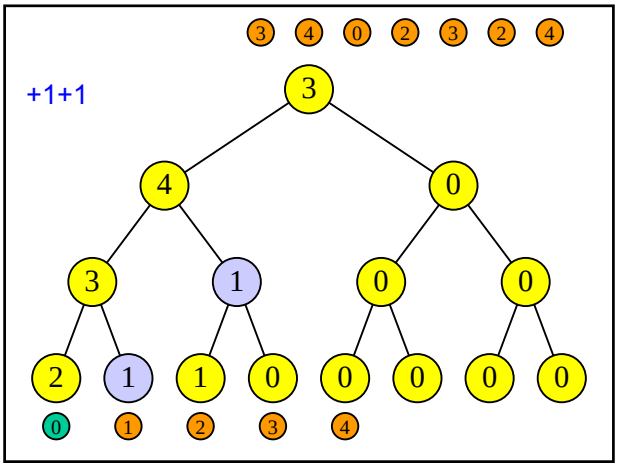

2 0 3 4 0 2 3 2 4
2
2 0
2 0 0 0
1 1 0 0 0 0 0 0
0 1 2 3 4

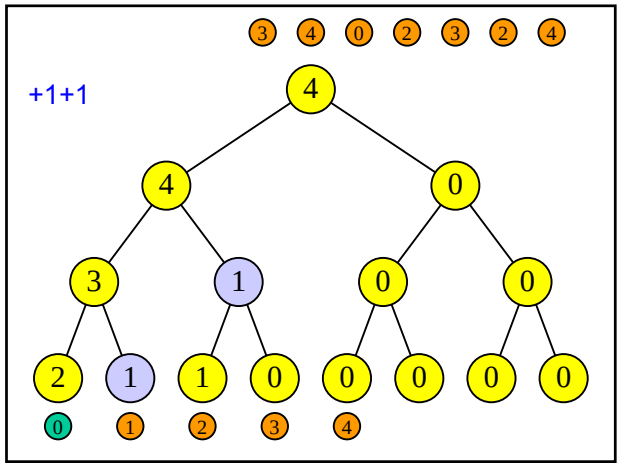

0 3 4 0 2 3 2 4
3
3 0
2 1 0 0
1 1 1 0 0 0 0 0
0 1 2 3 4

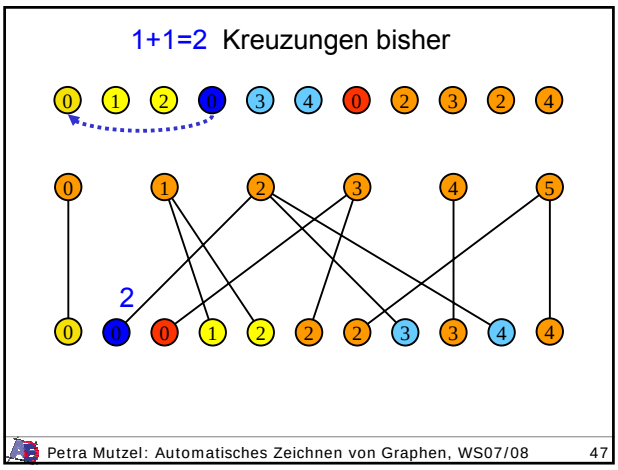

0 3 4 0 2 3 2 4
3
3 0
2 1 0 0
1 1 1 0 0 0 0 0
0 1 2 3 4

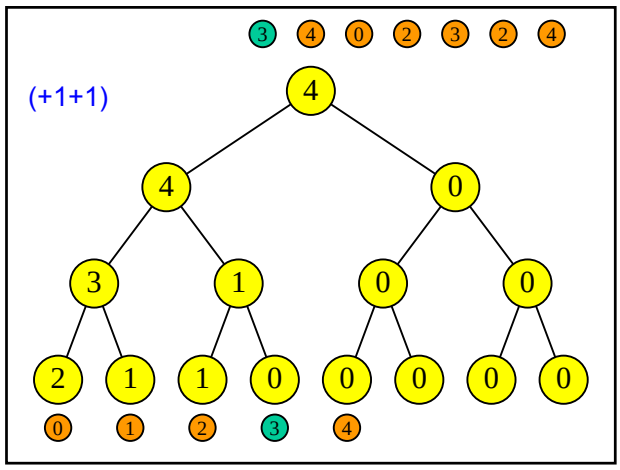

+1

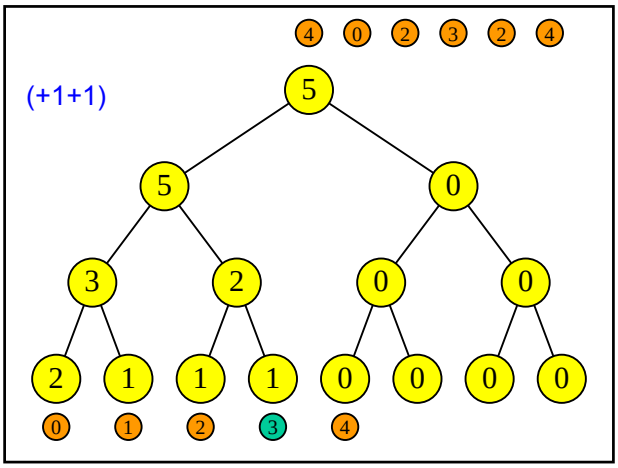

+1+1

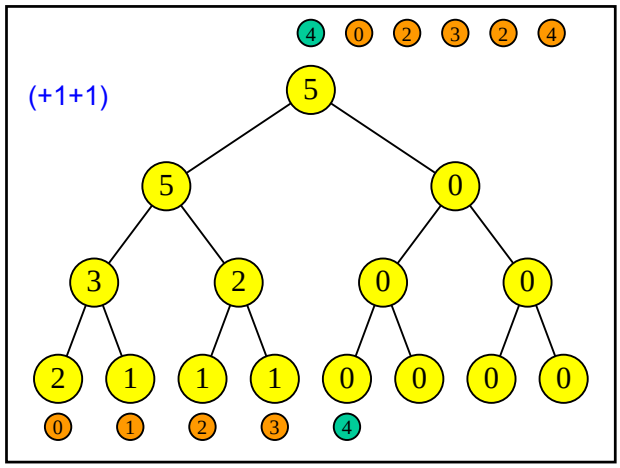

+1+1

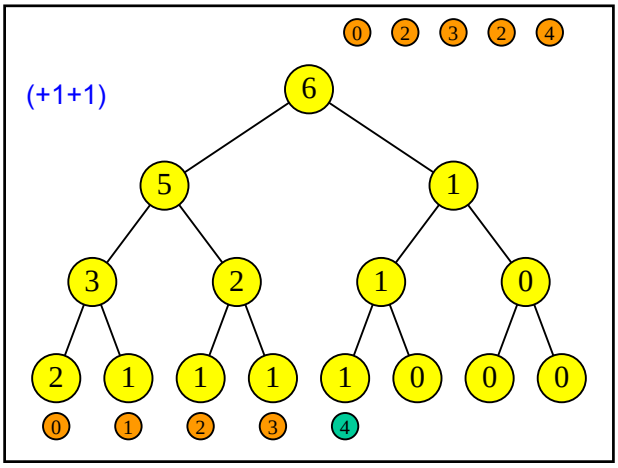

+1+1

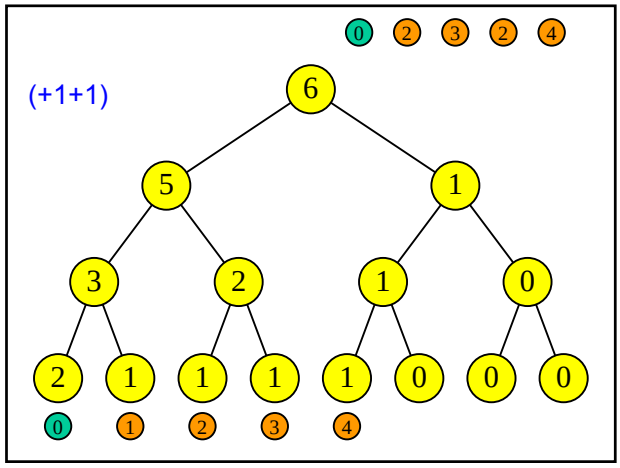

1+1=2 Kreuzungen bisher
2
Petra Mutzel: Automatisches Zeichnen von Graphen, WS07/08 47

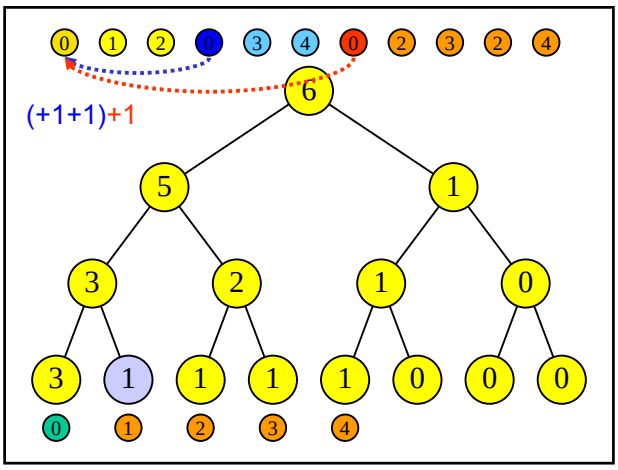

(+1+1)

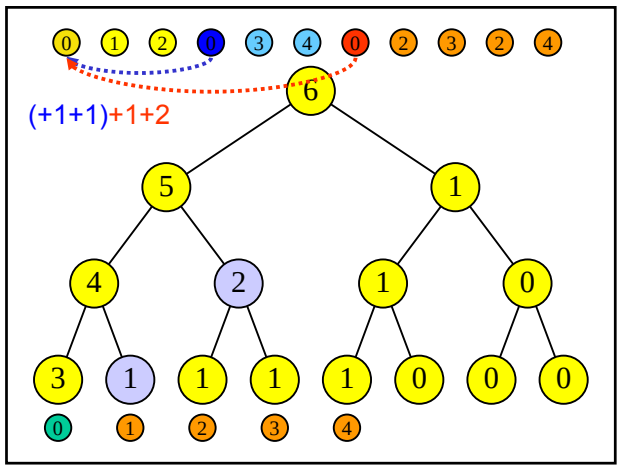

4 0 2 3 2 4
(+1+1)
5
5 0
3 2 0 0
2 1 1 1 0 0 0 0
0 1 2 3 4

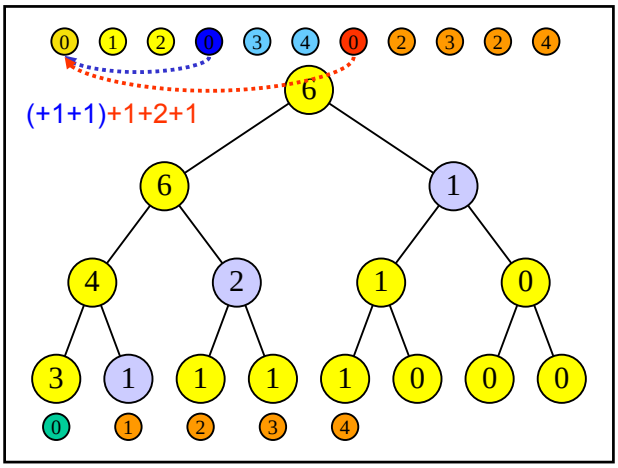

4 0 2 3 2 4
(+1+1)
5
5 0
3 2 0 0
2 1 1 1 0 0 0 0
0 1 2 3 4

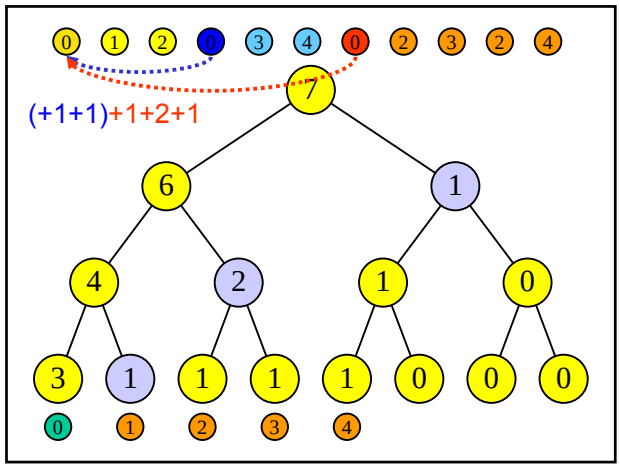

0 2 3 2 4
(+1+1)
6
5 1
3 2 1 0
2 1 1 1 1 0 0 0
0 1 2 3 4

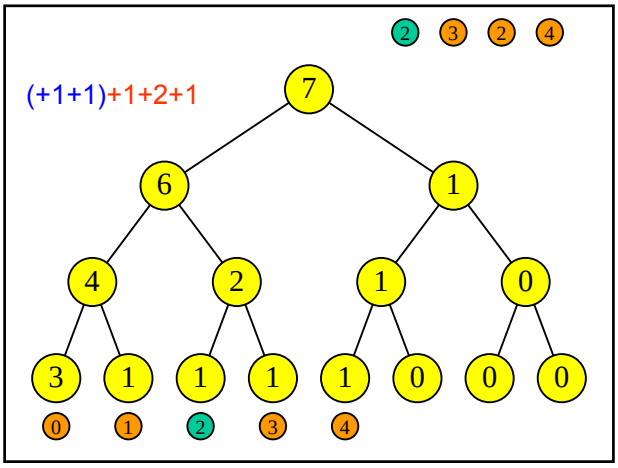

0 2 3 2 4
(+1+1)
6
5 1
3 2 1 0
2 1 1 1 1 0 0 0
0 1 2 3 4

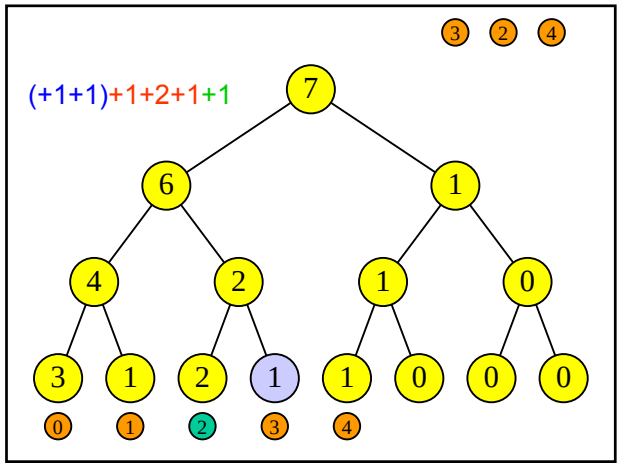

0 1 2 0 3 4 0 2 3 2 4
(+1+1)+1
6
5 1
3 2 1 0
3 1 1 1 1 0 0 0
0 1 2 3 4

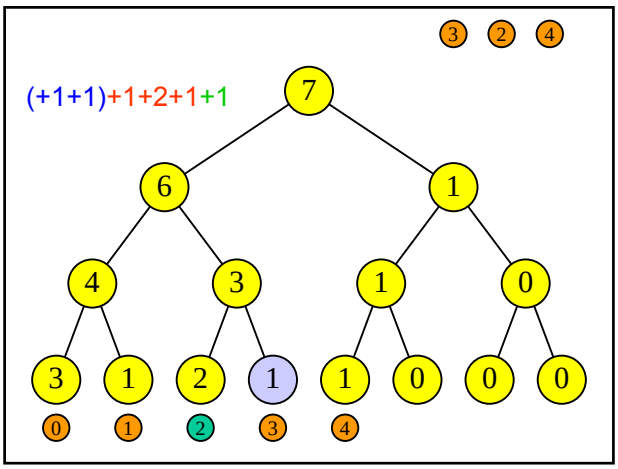

0 1 2 0 3 4 0 2 3 2 4
(+1+1)+1+2
6
5 1
4 2 1 0
3 1 1 1 1 0 0 0
0 1 2 3 4

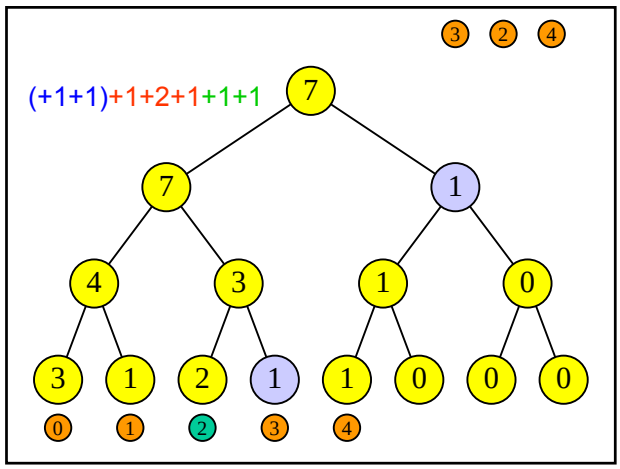

0 1 2 0 3 4 0 2 3 2 4
(+1+1)+1+2+1
6
6 1
4 2 1 0
3 1 1 1 1 0 0 0
0 1 2 3 4

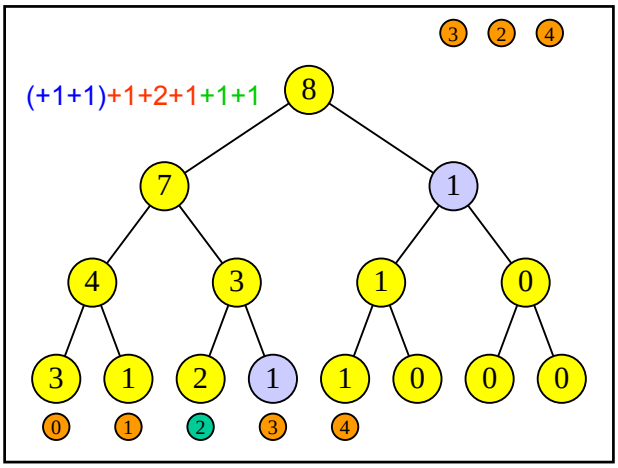

0 1 2 0 3 4 0 2 3 2 4
(+1+1)+1+2+1
7
6 1
4 2 1 0
3 1 1 1 1 0 0 0
0 1 2 3 4

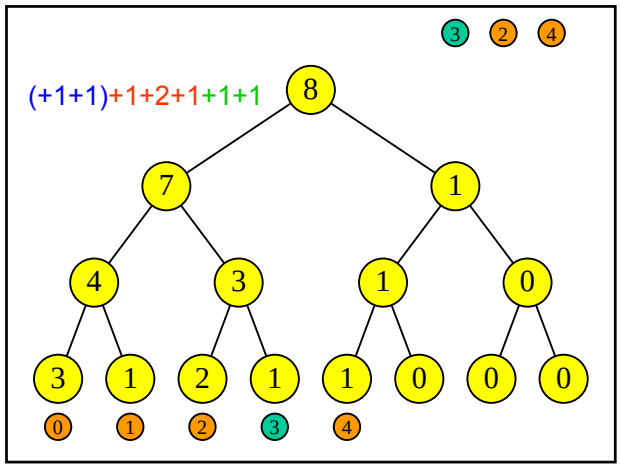

2 3 2 4
(+1+1)+1+2+1
7
6 1
4 2 1 0
3 1 1 1 1 0 0 0
0 1 2 3 4

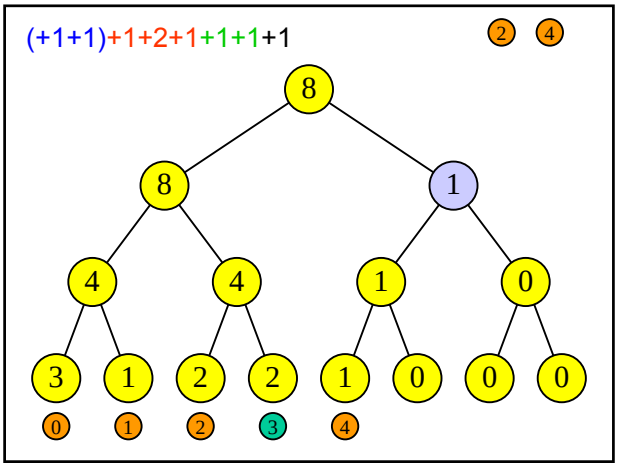

3 2 4
(+1+1)+1+2+1+1
7
6 1
4 2 1 0
3 1 2 1 1 0 0 0
0 1 2 3 4

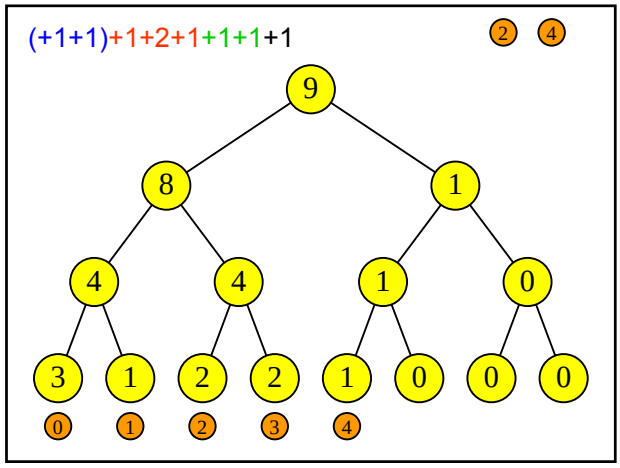

3 2 4
(+1+1)+1+2+1+1
7
6 1
4 3 1 0
3 1 2 1 1 0 0 0
0 1 2 3 4

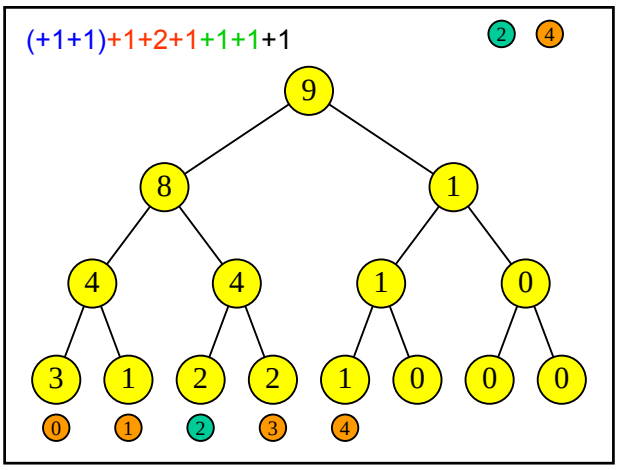

3 2 4
(+1+1)+1+2+1+1+1
7
7 1
4 3 1 0
3 1 2 1 1 0 0 0
0 1 2 3 4

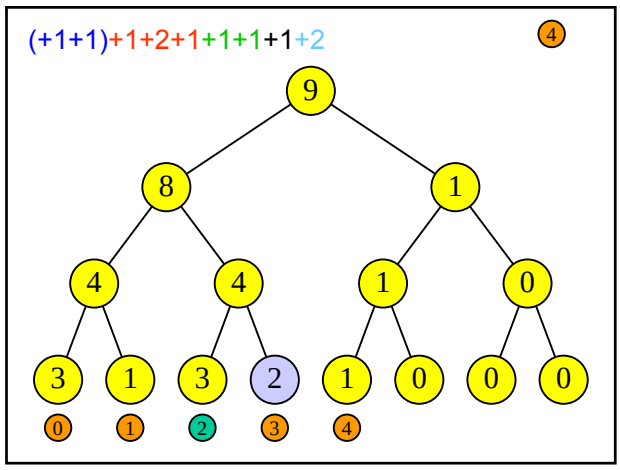

(+1+1)+1+2+1+1+1
8
7
1
4
3
1
0
3
1
2
1
1
0
0
0
0
1
2
3
4

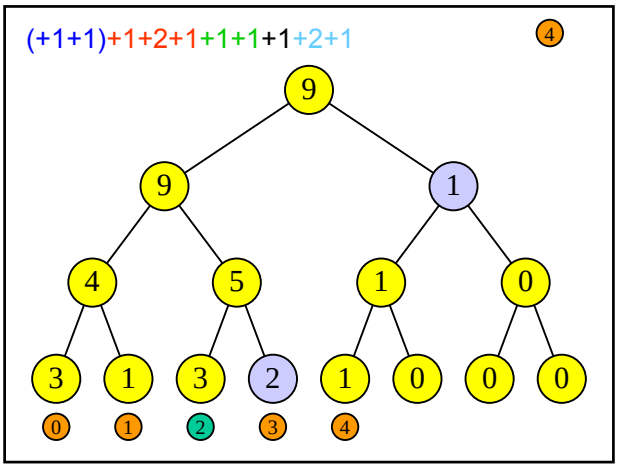

(+1+1)+1+2+1+1+1
8
7
1
4
3
1
0
3
1
2
1
1
0
0
0
0
1
2
3
4

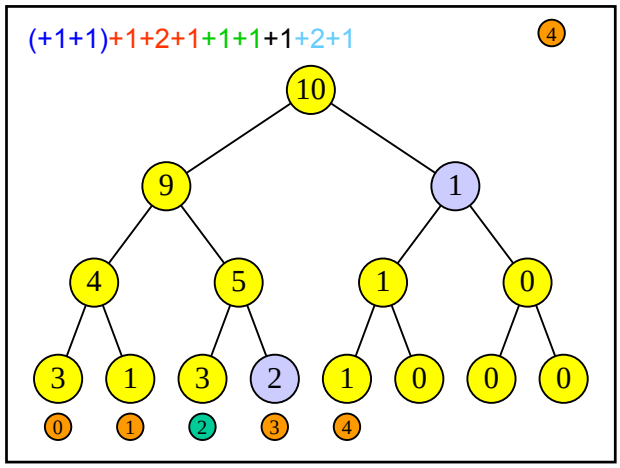

(+1+1)+1+2+1+1+1+1
8
8
1
4
4
1
0
3
1
2
2
1
0
0
0
0
1
2
3
4

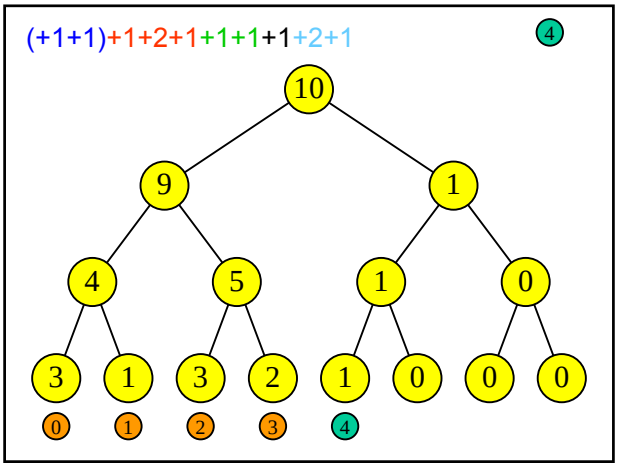

(+1+1)+1+2+1+1+1+1
9
8
1
4
4
1
0
3
1
2
2
1
0
0
0
0
1
2
3
4

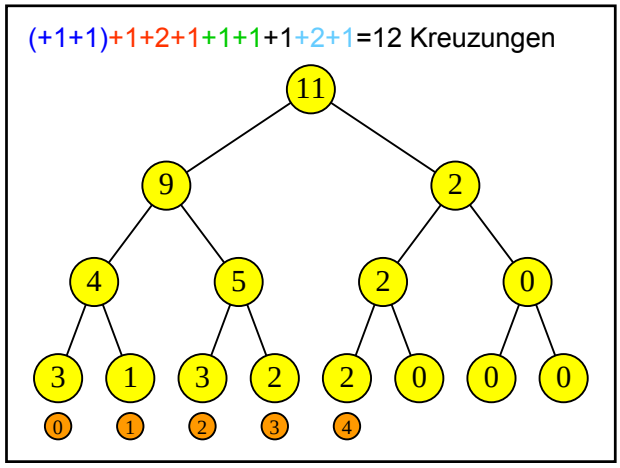

(+1+1)+1+2+1+1+1+1
9
8
1
4
4
1
0
3
1
2
2
1
0
0
0
0
1
2
3
4

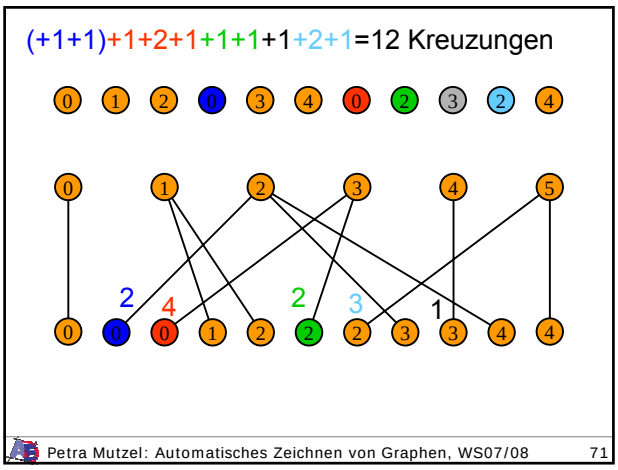

(+1+1)+1+2+1+1+1+1+2
9
8
1
4
4
1
0
3
1
3
2
1
0
0
0
0
1
2
3
4

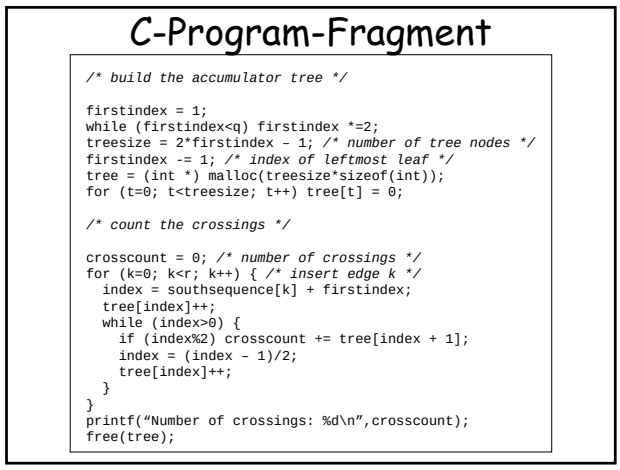

(+1+1)+1+2+1+1+1+1+2+1
4
9
9
1
4
5
1
0
3
1
3
2
1
0
0
0
0
1
2
3
4


(+1+1)+1+2+1+1+1+1+2+1
4
10
9
1
4
5
1
0
3
1
3
2
1
0
0
0
0
1
2
3
4


(+1+1)+1+2+1+1+1+1+2+1
4
10
9
1
4
5
1
0
3
1
3
2
1
0
0
0
0
1
2
3
4


(+1+1)+1+2+1+1+1+1+2+1=12 Kreuzungen
11
9
2
4
5
2
0
3
1
3
2
2
0
0
0
0
1
2
3
4


(+1+1)+1+2+1+1+1+1+2+1=12 Kreuzungen
0 1 2 0 3 4 0 2 3 2 4
0 1 2 3 4 5
2 4 2 3 1
0 0 0 1 2 2 2 3 3 4 4
Petra Mutzel: Automatisches Zeichnen von Graphen, WS07/08
71


C-Program-Fragment
/* build the accumulator tree */
firstindex = 1;
while (firstindex<q) firstindex *=2;
treesize = 2*firstindex - 1; /* number of tree nodes */
firstindex -= 1; /* index of leftmost leaf */
tree = (int *) malloc(treesize*sizeof(int));
for (t=0; t<treesize; t++) tree[t] = 0;
/* count the crossings */
crosscount = 0; /* number of crossings */
for (k=0; k<r; k++) { /* insert edge k */
index = southsequence[k] + firstindex;
tree[index]++;
while (index>0) {
if (index%2) crosscount += tree[index + 1];
index = (index - 1)/2;
tree[index]++;
}
}
printf("Number of crossings: %d\n",crosscount);
free(tree);

Algorithmus CrossCount

```
1. Sortiere die untere Schicht lexikographisch →
     southsequence[0..r-1]
2. Initialisierung des Accumulator Tree
3. CrossCount = 0;                /* Kreuzungsanzahl */
4. For (k=0; k<r; k++) {          /* füge Kante k ein */
5.    index = southsequence[k]
6.    tree[index]++;              /* Eintrag auf Blattebene */
7.    While (index>0) {           /* laufe Baum hoch */
8.       If (index zu linkem Kind gehörig)
9.          crosscount += tree[index + 1];
9.       Wandere eine Schicht nach oben
10.      tree[index]++;           /* Eintrag auf aktueller Schicht */
11.   }
12.}
```


Praktische Experimente

SAN Sander [1995] $O(|E| + |C|)$

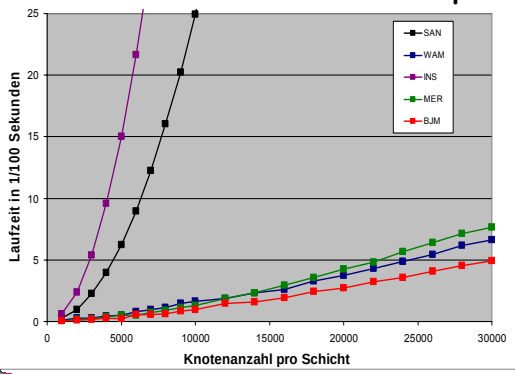
WAM Waddle & Malhotra [1999] $O(|E| \log |V|)$

INS Insertion Sort $O(|E| + |C|)$

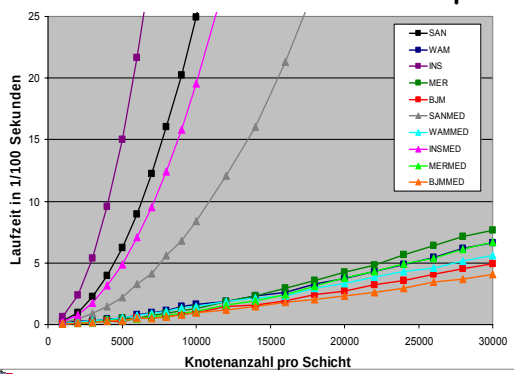
MER Merge Sort $O(|E| \log \text{run}(p))$

BJM Neuer Algorithmus $O(|E| \log |V_{small}|)$

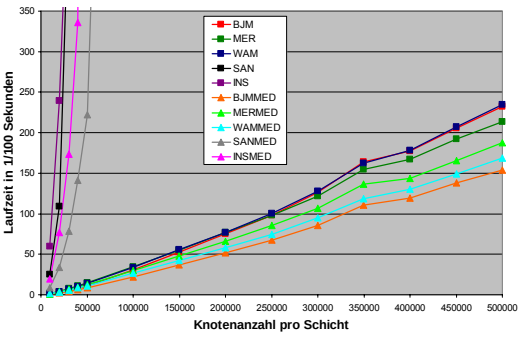
Laufzeit für dünne Graphen



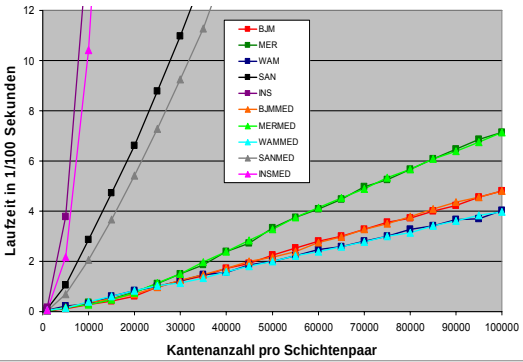
Laufzeit für dünne Graphen



Laufzeit für große dünne Graphen



Laufzeit für dichte Graphen



Computational Experiments

AT&T Graphen

1. Phase: Schichteneinteilung von AGD
- longest-path-layering: 30,061 Instanzen
- Coffman-Graham-layering: 57,300 Instanzen

Für jede Instanz:
10 zufällige Umsortierungen
gefolgt durch einen Median-Sortierschritt

Anzahl an Testläufen:
601,220 “longest path”
1,146,000 “Coffman-Graham”

Computational Experiments

AT&T Graphen

Longest path layering

1 bis 6,566 obere Knoten, 63 durchschnittlich
1 bis 5,755 untere Knoten, 57 durchschnittlich
1 bis 6,566 Kanten, 64 durchschnittlich

Zufällige Umsortierungen:
0 bis 10,155,835 Kreuzungen, 24,472 durchschnittlich

Median-sortierte Schichten:
0 bis 780,017 Kreuzungen, 182 durchschnittlich

Computational Experiments

AT&T Graphen

Coffman-Graham layering

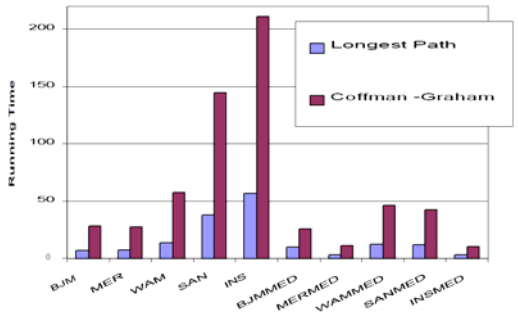
1 bis 3,278 obere Knoten, 142 durchschnittlich
1 bis 3,278 untere Knoten, 137 durchschnittlich
1 bis 3,276 Kanten, 141 durchschnittlich

Zufällige Umsortierungen:
0 bis 2,760,466 Kreuzungen, 47,559 durchschnittlich

Median-sortierte Schichten:
0 bis 2,872 Kreuzungen, 4 durchschnittlich

Graphen aus der Praxis

AT&T Graphen



Folgerung

WAM, MER, BJM
Reduzieren die Laufzeit
des Sugiyama-Algorithmus
signifikant!

BJM ist sehr einfach
zu implementieren!

Offenes Problem

Gegeben: eine Permutation von $0..n\text{-}1$:

Ist wirklich Zeit $\Theta(n \log n)$ nötig, um
die Anzahl der Inversionen zu zählen?

Vielen Dank