Institut für Computergraphik und Algorithmen

Abteilung für Algorithmen und Datenstrukturen

TECHNISCHE UNIVERSITÄT WIEN

Prof. Petra Mutzel
Gunnar Klau
Ivana Ljubic
René Weiskircher

Wintersemester 2000/2001

Klausur zur Vorlesung
Algorithmen und Datenstrukturen 2
19. Jänner 2000

a.) Machen Sie bitte die folgenden Angaben in deutlicher Blockschrift:

Name: ______________________ Vorname: ______________________

Matrikelnummer: ______________________ Studienkennzahl: ______________________

b.) Legen Sie während der Klausur Ihren Studentenausweis vor sich auf das Pult.

c.) Diese Klausur enthält Multiple–Choice–Fragen. Für jede dieser Aufgaben kann es eine oder mehrere richtige Lösungen geben. Markieren Sie diese **deutlich und unzweideutig** durch Ankreuzen. Alle Multiple–Choice–Aufgaben sind gleich viel wert; für jede richtige Antwort bekommen Sie Pluspunkte, für jedes falsche Kreuz gibt es Minuspunkte.

d.) Denken Sie daran, dass keinerlei Hilfsmittel erlaubt sind – weder Taschenrechner, irgendwelche Unterlagen, Handys,...

VOR DER ABGABE AUSZUFÜLLEN:

e.) Geben Sie bitte die Anzahl der zusätzlich abgegebenen Blätter an: __________

f.) Kreuzen Sie bitte die von Ihnen bearbeiteten Aufgaben in der ersten Zeile der Tabelle an:

Aufgabe	A 1	A 2	A 3	A 4	A 5	A 6		Note
bearbeitet							—	——
maximale Punktzahl	6	6	6	10	6	16	50	——
erreichte Punktzahl								

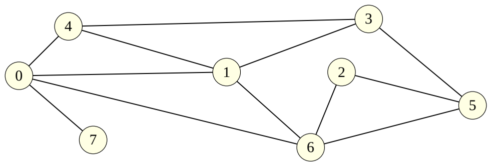
Viel Erfolg!

Aufgabe 1: Depth-First-Search **(6 Punkte)**

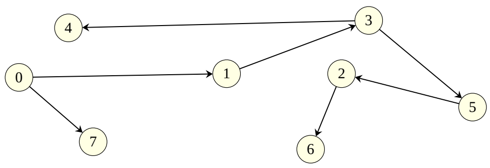
Gegeben sei ein Graph $G = (V, E)$, und ein zugehöriger Depth-First-Search Baum. Welche Kreise wurden bei der Durchführung des Depth-First-Search Algorithmus gefunden?

input: Graph $G = (V, E)$, Wurzel s;
output: Kreise in G;
MAIN
begin
$Stack := \emptyset$;
$visited[v] := False, \forall v \in V$;
DFSCYCLE($s, \emptyset, Stack$);
end

DFSCYCLE($v, p, Stack$)
begin
$visited[v] := True$;
$Stack := Stack \cup \{v\}$;
for all $(v, w) \in E$ **do**
  if $w \neq p$ **then**
    if $visited[w]$ und $w \in Stack$ **then**
      Drucke alle Knoten die in $Stack$ zwischen v und w liegen;
    else
      DFSCYCLE($w, v, Stack$);
    end if
  end if
end for
$Stack := Stack \setminus \{v\}$;
end



Graph $G = (V, E)$



DFS–Baum

◯ a) 0,4,3,5,2,6 ◯ b) 1,3,5,6 ◯ c) 0,1,6 ◯ d) 4,3,1,0

Aufgabe 2: Evolutionäre Algorithmen **(6 Punkte)**
Gegegben sei ein evolutionärer Algorithmus für das Map Labeling Problem. Jede Lösung ist durch einen Vektor $x = (x_1, \ldots, x_n)$ repräsentiert, wobei Komponente x_i zu dem Label mit Nummer i gehört. Es gilt $x_i > 0$, falls das Label in der Lösung ohne Überlappungen platziert ist und $x_i = 0$ sonst. Der Algorithmus erzeugt nur Lösungen ohne Überlappungen.
Welche Formel kann als *Fitness-Funktion* in einem evolutionären Algorithmus verwendet werden, wenn Fitness-proportionale Selektion verwendet wird?

◯ $f(x) = \frac{1}{\sum_{i=1}^{n} \operatorname{sgn} x_i}$

◯ $f(x) = -\sum_{i=1}^{n} \operatorname{sgn} x_i$

◯ $f(x) = \sum_{i=1}^{n} (1 - \operatorname{sgn} x_i)$

◯ $f(x) = -\sum_{i=1}^{n} (1 - \operatorname{sgn} x_i)$

Aufgabe 3: Stabile Menge **(6 Punkte)**
Für einen Graphen $G = (V, E)$ ist eine *stabile Menge* eine Teilmenge V' von Knoten, so dass für jede Kante e in E höchstens ein Endknoten in V' enthalten ist. Genauer

$$\forall e = (v, w) \in E : v \notin V' \text{ oder } w \notin V'.$$

Eine interessantes (und übrigens auch NP-schweres) Problem ist die Suche nach einer größten stabilen Menge. Gegeben sei folgender Algorithmus für dieses Problem.

input: Graph $G = (V, E)$;
output: stabile Menge V' in G;
$V' := \emptyset$;
$U := V$;
while U ist nicht leer **do**
  $x :=$ Knoten mit minimalem Grad in dem durch U induzierten Graphen;
  $V' := V' \cup \{x\}$;
  Lösche x und alle seine Nachbarn aus U;
end while
return V';

Es handelt sich hierbei um

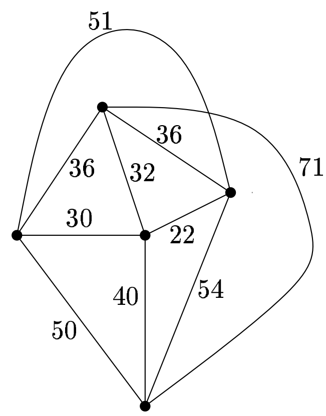
◯ einen Greedy–Algorithmus.

◯ eine konstruktive Heuristik.

◯ einen exakten Algorithmus.

◯ eine Verbesserungsheuristik.

Aufgabe 4: Travelling–Salesman–Problem (TSP) **(10 Punkte)**
Wenden Sie die Christophides–Heuristik auf den folgenden Beispielgraphen an. Veranschaulichen Sie in Ihrem eigenen Interesse **alle** Schritte, die zur Konstruktion einer Tour führen.



Aufgabe 5: Suchen in Texten **(6 Punkte)**

Sie möchten in einem Text, der im Array $T[1], \ldots, T[N]$ gespeichert ist, das Muster

CTCGGCACTGCTCTCGGCT

suchen, das im Array $P[1], \ldots, P[M]$ gespeichert ist. Dazu verwenden Sie den Algorithmus von Knuth-Morris-Pratt in der folgenden Realisierung:

(0) Algorithmus Knuth-Morris-Pratt(T,P);
(1) Berechnung des Arrays `NEXT[]`;
(2) $j = 0$;
(3) Für $i = 1, ..., N$ {
(4)   Solange $((j > 0)$ UND $(P[j+1]! = T[i]))$ { $j =$ `NEXT`$[j]$;}
(5)   Falls $(P[j+1] == T[i])$ { $j = j + 1$;}
(6)   Falls $(j == M)$ { P gefunden; STOP;}
(7) }

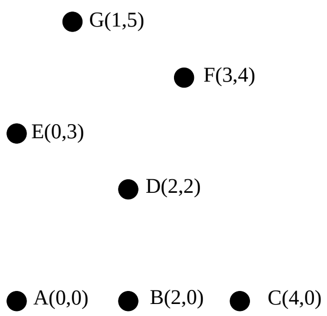
Es wird also zunächst in Schritt 1 das Array `NEXT[]` berechnet.
Geben Sie das `NEXT`-Array für diesen Fall nach der Berechnung an:

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
`NEXT`																			

Aufgabe 6: Scanline **(16 Punkte)**

Wir betrachten eine Menge von n Punkten in der Ebene. Für zwei Punkte $x = (x_1, x_2)$ und $y = (y_1, y_2)$ sagen wir x *dominiert* y falls $x_1 \geq y_1$ und $x_2 \geq y_2$ gilt. Ein Punkt ist *maximal*, wenn er von keinem anderen Punkt dominiert wird.

a.) Geben Sie die Menge aller maximalen Punkte in der folgenden Zeichnung an (die Koordinaten stehen neben den Namen der Punkte in Klammern).



Beispielmenge von Punkten

b.) Geben Sie einen Scanline-Algorithmus in Pseudocode an, um die Menge der maximalen Punkte aus einer Menge mit n Punkten zu bestimmen (Hinweis: Scan von rechts).

c.) Wie ist die Laufzeit ihres Algorithmus in O-Notation (Begründung)?