Algorithmen und Programmierung 3, WS 2019/2020 — 15. Übungsblatt

ohne Abgabe

---

101. Dynamische Programmierung zur Anzahlbestimmung

Für zwei Folgen $(a_1, \ldots, a_m)$ und $(b_1, \ldots, b_n)$ und eine gegebene Länge k soll die *Anzahl* der Paare (i, j) bestimmt werden, sodass the Teilwörter $(a_i, \ldots, a_{i+k-1})$ und $(b_j, \ldots, b_{j+k-1})$ übereinstimmen.

Der triviale Algorithmus, der das Problem genauso löst, wie es formuliert ist, macht $(m-k+1)(n-k+1)k \leq mnk$ Vergleiche und benötigt neben der Eingabe und der Ausgabe nur konstanten zusätzlichen Speicher.

Entwerfen Sie einen effizienteren Algorithmus. Schreiben Sie dafür ein Programm in Pseudocode. Analysieren Sie die Laufzeit und den Speicherbedarf Ihres Algorithmus.

Sie können das Problem durch dynamisches Programmieren lösen. (Es gibt auch alternative Ansätze mit Hilfe von Suffixbäumen.)

102. Kürzeste Wege

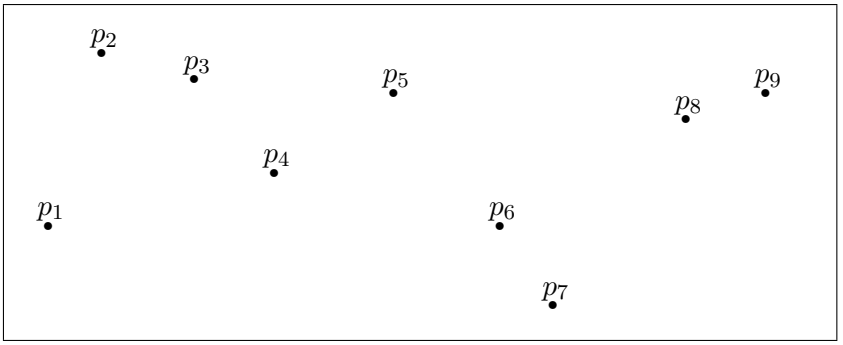
(a) Finde sie mit dem Algorithmus von Dantzig die kürzesten Wege zwischen allen Paaren von Knoten im Graphen mit der folgenden Distanzmatrix:

$$C = \begin{pmatrix} - & 30 & 12 & 5 & 2 \\ \infty & - & 17 & -4 & 3 \\ 11 & -6 & - & 11 & \infty \\ -2 & 7 & \infty & - & 6 \\ 3 & 1 & 6 & \infty & - \end{pmatrix}$$

(b) Zeigen Sie: Ein Graph hat genau dann einen negativen Kreis, wenn es $i < k$ gibt mit $A^k_{ik} + A^k_{ki} < 0$. Hier ist A^k_{ij} die Länge des kürzesten Weges von v_i nach v_j im Teilgraphen, der aus den Knoten $\{v_1, \ldots, v_k\}$ besteht.

103. Konvexe Hülle

Bestimmen Sie die untere konvexe Hülle der sortierten Punktmenge $p_1, \ldots, p_9$ mit dem inkrementellen Algorithmus aus der Vorlesung. Geben Sie die Folge der Orientierungstests an, die der Algorithmus ausführt.

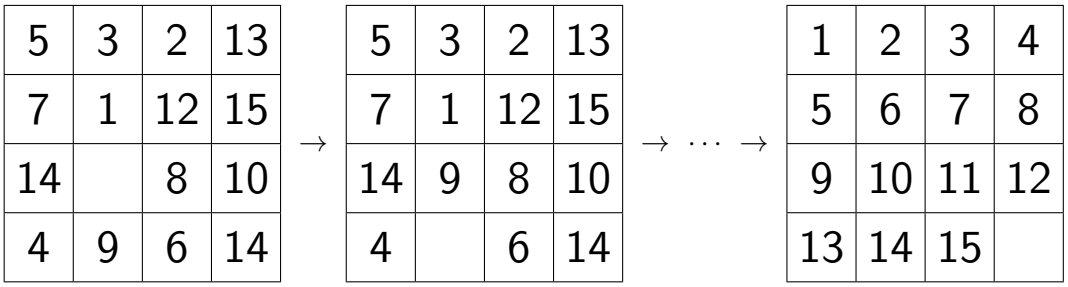


104. Mittlere innere Weglänge

Schreiben Sie ein Programm in Java, Haskell, oder Python, das für einen gegebenen binären Suchbaum die erwartete Anzahl von Vergleichen, die zum Finden eines zufällig ausgewählten Knotens in dem Baum im Durchschnitt erforderlich sind. (Jeder Knoten wird mit gleicher Wahrscheinlichkeit gesucht.)

105. Wege in Graphen

Das berühmte 14-15-Spiel besteht aus 15 nummerierten quadratischen Blöcken, die auf einem 4×4-Spielfeld beweglich angeordnet sind. Eines der 16 Felder ist frei. Ein Zug besteht darin, dass man einen Stein von einem benachbarten Feld auf das freie Feld verschiebt. Das Ziel ist, von einer gegebenen Ausgangsposition ausgehend, die Steine in die sortierte Reihenfolge zu bringen, sodass das freie Feld rechts unten ist.



Diskutieren Sie, wie man dieses Problem als Wegeproblem in einem Graphen modellieren kann. Wie viele Knoten und wie viele Kanten hat der Graph? Ist der Graph gerichtet oder ungerichtet? Welche Schwierigkeiten können bei diesem Problem auftreten?

106. Implementierung eines abstrakten Datentyps

Die folgende Klasse `BitSet` implementiert einen Menge von (nicht zu großen) nichtnegativen ganzen Zahlen als Bitvektor, gruppiert in 64-Bit-Wörter.

```
public class BitSet {
  int maxvalue;
  long A[];

  BitSet(int maxv) {
     maxvalue = maxv;
     int size = 1+maxv/64;
     A = new long[size];
  }

  public boolean contains(int i) {
     if ((i>maxvalue) || (i<0))
                     return false;
     int k = i/64;
     int shift = i-k*64;
     return ((A[k]>>shift) & 1)==1;
  }
  ...
```

Python-Version:

```
class BitSet:

  def __init__(self, maxv):
     self.maxvalue = maxv
     size = 1+maxv//64
     self.A = [0]*size


  def contains(self, i):
     if i>self.maxvalue or i<0:
        return False
     k = i//64
     shift = i%64
     return (self.A[k]>>shift) & 1

  ...
```

(a) Definieren Sie einen passenden abstrakten Datentyp, der auch die Methoden `add` und `remove` enthält.

(b) Ergänzen Sie die Implementierung der Methoden `add` und `remove` in Java oder Python.

(c) Geben Sie die Abstraktionsfunktion und gegebenfalls die Darstellungsinvarianten und die Nebenbedingungen dieser Implementierung an.

(d) Beweisen Sie die Korrektheit der Implementierung.

$A^k_{kj} = \min\{c_{ki} + A^{k-1}_{ij} \mid 1 \le i \le k-1\}$

$A^k_{ik} = \min\{A^{k-1}_{ij} + c_{jk} \mid 1 \le j \le k-1\}$

$A^k_{ij} = \min\{A^{k-1}_{ij}, A^k_{ik} + A^k_{kj}\}$

$A^k_{ii} = 0$

A^1	1
1	0

A^2	1	2
1	0	$A^2_{1,2}$ 30
2	$A^2_{2,1}$ ∞	0

$A^2_{1,2} = \min\{A^1_{11} + c_{12}\}$ = 0 + 30

$A^2_{2,1} = \min\{c_{21} + A^1_{11}\}$ = ∞ + 0

A^3	1	2	3
1	0	A^3_{12} 6	12
2		0	
3		-6	0

$\min\{c_{31} + A^2_{12}, c_{32} + A^2_{22}\}$ (11, 30, -6, 0)

$A^3_{1,2} = \min\{A^2_{12}, A^3_{13} + A^3_{32}\} = \min\{30, 12-6\}$

30, A^2_{11} (0) + c_{13} (12) | A^2_{12} (30) + c_{23} (17) → 12

Editier Abstand $a_1, \dots, a_m$ → $b_1, \dots, b_n$

a_i	A	M	O	N
b	A	V	O	X

1. Teilprobleme:

$a_1, \dots, a_i \rightarrow b_1, \dots, b_j \quad = E_{ij}$

$1 \le i \le m \qquad 1 \le j \le n$

Rekursionsgleichung

$E_{ij} = \min \begin{cases} E_{i-1,j-1} & a_i = b_j \quad \text{unverändert} \\ E_{i-1,j-1}+1 & a_i \ne b_j \quad \text{Ersetzen} \\ E_{i-1,j}+1 & \text{Löschen von } a_i \\ E_{i,j-1}+1 & \text{Einfügen von } b_j \end{cases}$

Randbedingung

$E_{i0} = i$ (Löschen) $\qquad E_{0j} = j$ (Einfügen)

AMON / AVOXE min:
- AMO / AVOX +1
- → AMO / AVOXE +1
- ↳ (AMONE / AVOXE) +1 = AMON / AVOX +1

E_{ij}	0	A	V	O	X	Z
0	0	1	2	3	4	5
A	1	E_{11} 0	1	2	3	4
M	2	1	1	2	3	4
O	3	2	2	1 (0+0)	2	3
N	4	3	3	2	2	3

$E_{11} = \min\{E_{00}, E_{10}, E_{01}\}$