Johdatus tekoälyyn, syksy 2011, Teemu Roos Harjoitus 1: Esimerkkiratkaisuja

Tehtävä 1. Tekoälytutkimus vs. sci-fi

- "paljon eri aloja ja haaroja", "detaljitasolla"
- artikkelien lukeminen haastavaa (monet kirjoitettu huonosti!)
- ei juurikaan pohdita mitä älykkyys tai tietoisuus on
- sovelletaan monenlaista matematiikkaa (diskreettiä, analyysia, lineaarialgebraa, todennäköisyyslaskentaa), tilastotiedettä, tietojenkäsittelytiedettä (laskennan mallit, algoritmien suunnittelu ja analyysi, ohjelmointia).
- Skynet ei näköpiirissä

Tehtävä 2. Turingin testi

CAPTCHA = "Completely Automated Public Turing test to tell Computers and Humans Apart"

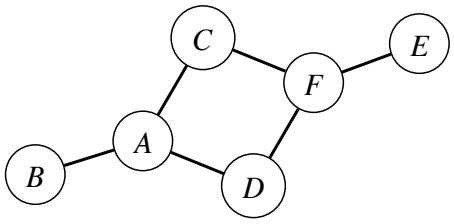
Esitetään ongelma, jonka ratkaiseminen on ihmiselle helppoa, mutta tietokoneelle ei. Usein kuva, jossa vääristettyä tekstiä, tms.

Amazonin Mechanical Turk on palvelu, jossa rutiinitehtäviä voi teettää halpatyövoimalla. Roskapostiohjelma voi esittää ihmistä ja läpäistä Turingin testin lähettämällä CAPTCHA-ongelmia ihmisten ratkottavaksi. Tuhannen CAPTCHA:n ratkomisesta saa tyypillisesti noin 0.80–1.20 dollaria palkkiota. Turingin testissä blogiohjelmisto on testaaja ja spammiohjelmisto yhdessä ihmisratkojien kanssa testattava.

Tehtävä 3. Etsintä: leveys- ja syvyyssuuntainen haku

Leveyssuuntainen läpikäynti:

```
A->B->C->D->F->E
```




```
[A]
A
[B,C,D]
B
[C,D]
C
[D,F]
D
[F]
F
[E]
E
[]
```

Syvyyssuuntainen läpikäynti:

```
A->D->F->E->C->B
```

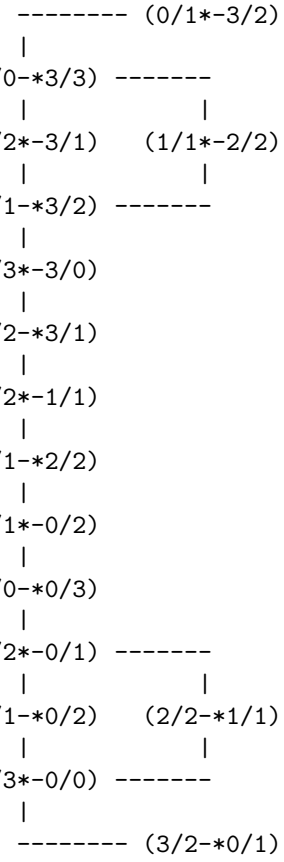
```
[A]
A
[B,C,D]
D
[B,C,F]
F
[B,C,E]
E
[B,C]
C
[B]
B
[]
```

Tehtävä 4. Etsintä ongelmanratkaisuna

Tilaesitys:

```
(läh./kann.-läh./kann.)  # vene='*'
```

Tilasiirtymäkaavio:



Tehtävät 5–6.

Python-esimerkki (kiitokset Ilari Sanille):

```
class Node:

        # state is any python object describing node state
        # for simple nodes, it can be a string
        def __init__(self, state):
                self.state = state
                self.edges = []
                self.previous = None

        def add_neighbor(self, node):
                self.edges.append(node)

        def neighbors(self):
                return self.edges

class Searcher:

        # start and goal must be Node objects
        # mode must be string "width" or "depth"
        def search(self, start, goal, mode):
                nodes = [start]
                visited = []
                while len(nodes) > 0:

                        # Choose the next node to visit
                        if mode == "width":
                                node = nodes.pop(0) # Treat as queue
                        elif mode == "depth":
                                node = nodes.pop() # Treat as stack
                        else:
                                raise Exception("Invalid mode")

                        if node is goal:
                                return node

                        # Add all unvisited neighbors
                        for neighbor in node.neighbors():
                                if neighbor not in visited:
                                        # Mark route to starting node
                                        neighbor.previous = node
                                        nodes.append(neighbor)

                        visited.append(node)

                # If we get here, a path to goal wasn't found
                return None


# Test script
def main():
```

```
        # Populate the graph
        a = Node("A")
        b = Node("B")
        c = Node("C")
        d = Node("D")
        e = Node("E")
        f = Node("F")
        a.add_neighbor(b)
        a.add_neighbor(c)
        a.add_neighbor(d)
        b.add_neighbor(a)
        c.add_neighbor(a)
        c.add_neighbor(f)
        d.add_neighbor(a)
        d.add_neighbor(f)
        e.add_neighbor(f)
        f.add_neighbor(c)
        f.add_neighbor(d)
        f.add_neighbor(e)

        # Search for path
        searcher = Searcher()
        node = searcher.search(a, f, "depth")

        # Print out path (in reverse order)
        while node:
                print node.state
                node = node.previous

if __name__ == "__main__":
    main()
```

Koeajo:

```
$ python harj1.py
F
D
A
```

Java-esimerkkit (kiitokset Antero Hynyselle):

```
/**
 * Class Solmu
 * Represents one state in a "searching problem" info structure.
 **/
class Solmu {
        Object info = null; // for storing "whatever" info object..
        Solmu[] neighbours = null;        // for the graph exercise
        boolean added = false;                // for the graph exercise

        Solmu() {
        }
```

```
        Solmu(Object o) {
                Solmu[] neighbours = null;
                boolean added = false;
                this.info = o;
        }

        void clean() {
                this.added        = false;
        }

        void setNeighbours(Solmu[] l) {
                this.neighbours = l;
        }

        void printNeighbours() {
                int i = 0;
                Solmu n;
                while (this.neighbours != null && this.neighbours.length > i) {
                        n = this.neighbours[i];
                        System.out.print(n.info);
                        i++;
                        if (this.neighbours.length > i) {
                                System.out.print(":");
                        }
                }
        }
}


/**
 * Class Jono
 * Represents bidirectional linked list for storing object info.
 **/
class Jono {
        private LineItem first;
        private LineItem last;
        private int size;

        // Create new list with default values.
        Jono() {
                this.first = null;
                this.last = null;
                this.size = 0;
        }

        /**
         * Returns last knot of this list.
         * @return LineItem
         **/
        public LineItem getLast() {
                return this.last;
        }
```

```
/**
 * Returns size of this list.
 * @return integer
 **/
public int getSize() {
        return this.size;
}

/**
 * Adds new knot in the beginning of the list.
 **/
public void addFirst(Object obj) {
        // Create new knot, populate it with info and set it as list first.
        LineItem k = new LineItem();
        k.data = obj;
        k.next = this.first;
        k.prev = null;
        this.first = k;

        // If next knot is found then set it pointing to current one.
        if (this.first.next != null) {
                this.first.next.prev = this.first;
        } else  { // Else this is also the last of the list...
                this.last = k;
        }
        this.size++;
}

/**
 * Adds new Solmu in the beginning of the list.
 * NOTE: will cause problems if same Solmu is in multiple Jono's
 * @param Solmu s
 */
public void addFirstSolmu(Solmu s) {
        LineItem item = new LineItem();
        item.data = s;
        item.next = this.first;
        item.prev = null;
        this.first = item;

        // If next knot is found then set it pointing to current one.
        if (this.first.next != null) {
                this.first.next.prev = this.first;
        } else  { // Else this is also the last of the list...
                this.last = item;
        }

        this.size++;
}

/**
 * Adds new knot in the end of the list.
```

```
 * NOTE: will cause problems if same Solmu is in multiple Jono's         *
 * @param Solmu s
 */
public void addLastSolmu(Solmu s) {
        LineItem item = new LineItem();
        item.data = s;

        // If this is the first knot added to list then set it as list first.
        if (this.last == null) {
                this.first = item;
        } else {
                item.prev = this.last;
                this.last.next = item;
        }

        // Set this object as list first and increase list size counter by one.
        this.last = item;
        this.size++;
}

/**
 * Removes knot from then end of the list returning related info object.
 * If list is empty then will return null.
 *
 * @return Object
 */
public Object removeLast() {
        // Make sure that list is not empty.
        if (this.last == null) {
                return null;
        }

        // Fetch info from last knot and set previous knot as list last.
        Object obj = this.last.data;

        // If this is the first knot of the list then set first as null.
        if (this.last.prev == null) {
                this.first = null;
        } else {
                this.last.prev.next = null;
        }
        this.last = this.last.prev;

        // NOTE: JAVA garbage collector takes care of releasing memory used by any "removed" k

        this.size--; // After knot is removed from the list decrease list size counter by one.

        return obj;
}

/**
 * Removes Solmu from the start of the list returning it.
 * If list is empty then will return null.
```

```
 * NOTE: will cause problems if same Solmu is in multiple Jono's
 * @return Solmu
 */
public Solmu removeFirstSolmu() {
        // Make sure that list is not empty.
        if (this.first == null) {
                return null;
        }

        // Remove first Solmu and set the next as the first.
        Solmu s = (Solmu) this.first.data;

        // If this is the first Solmu of the list then set first as null.
        if (this.first.next == null) {
                this.last = null;
        } else {
                this.first.next.prev = null;
        }
        this.first = this.first.next;

        // NOTE: JAVA garbage collector takes care of releasing memory used by any "removed" S

        this.size--; // After Solmu is removed from the list decrease list size counter by one

        return s;
}

/**
 * Removes Solmu from then end of the list returning it.
 * If list is empty then will return null.
 * NOTE: will cause problem's if the same Solmu is in multiple Jono's
 * @return Solmu
 */
public Solmu removeLastSolmu() {
        // Make sure that list is not empty.
        if (this.last == null) {
                return null;
        }

        // Fetch info from first Solmu and set previous as first of the list.
        Solmu s = (Solmu) this.last.data;

        // If this is the first of the list then set first as null.
        if (this.last.prev == null) {
                this.first = null;
        } else {
                this.last.prev.next = null;
        }
        this.last = this.last.prev;

        this.size--; // After Solmu is removed from the list decrease list size counter by one

        return s;
```

```
        }

        /**
         * Prints info of knots stored in list.
         **/
        public void printContent() {
                LineItem tmp        = this.first;
                int i                       = 0;
                while (tmp != null && this.size > i) {
                        if (tmp.data.getClass().getSimpleName().equals("Solmu")) {
                                Solmu s = (Solmu) tmp.data;
                                System.out.print("[" + s.info + "]");
                                if (s.neighbours != null && s.neighbours.length > 0) {
                                        System.out.print("(");
                                        s.printNeighbours();
                                        System.out.print(") ");                                 }
                        } else {
                                System.out.print("[" + tmp.data + "] ");
                        }
                        i++;
                        tmp = tmp.next;
                }
                System.out.println("");        // line breaker..
        }
}

/**
 * Class LineItem
 * Represents one item in Stack
 **/
class LineItem {
        Object data                = null;
    LineItem next        = null;
    LineItem prev        = null;
}


/**
 * Class Pino
 * Represents stack for storing object info.
 **/
class Pino {
        private StackItem top;
        private int size;

        // Create new Pino with default values.
        Pino() {
                this.top = null;
                this.size = 0;
        }

        /**
         * Returns and removes top item of the stack.
```

```
 * @return StackItem
 **/
public StackItem pop() {
        StackItem item = this.top;
        this.top = item.below;
        item.below = null;
        this.size--;
        return item;
}

/**
 * Adds item on top of the stack.
 * @param Object
 **/
public void push(Object data) {
        StackItem item = new StackItem();
        item.data = data;
        item.below = this.top;
        this.top = item;
        this.size++;
}

/**
 * Returns size of this Pino.
 * @return integer
 **/
public int getSize() {
        return this.size;
}

/**
 * Prints info of knots stored in Pino.
 **/
public void printContent() {
        StackItem tmp         = this.top;
        int i                 = 0;
        while (tmp != null && this.size > i) {
                if (tmp.data.getClass().getSimpleName().equals("Solmu")) {
                        Solmu s = (Solmu) tmp.data;
                        System.out.print(s.info);
                        if (s.neighbours != null && s.neighbours.length > 0) {
                                System.out.print("(");
                                s.printNeighbours();
                                System.out.print(")");
                        }
                } else {
                        System.out.print(tmp.data);
                }
                System.out.print(" | ");
                i++;
                tmp = tmp.below;
        }
        System.out.println("");        // change line..
```

```
        }
}

/**
 * Class StackItem
 * Represents one item in Stack
 **/
class StackItem {
        Object data = null; // for storing "whatever" info object..
    StackItem below = null; // for the stack exercise
}


/**
 * AI Exercise main class for executing various "introduction to AI" exercise codes..
 * @author anteroh
 */
public class AIExercise1 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        int size = 3;

                String exercise = "";

                if (args.length > 0) {
                    try {
                            exercise = args[0].toLowerCase();
                        } catch (Exception e) {
                                exercise = "";
                        }
                }

                if (exercise.equals("pino")) {
                        System.out.println("Lisätään pinoon yksi kerrallaan");
                        Pino p = new Pino();
                        for (int i = 0; i < size; i++) {
                                Object o = (Object) (i + 1);
                                p.push(o);
                                p.printContent();
                        }
                        System.out.println("Poistetaan pinosta yksi kerrallaan");
                        for (int i = 0; i < size; i++) {
                                Object o = p.pop();
                                p.printContent();
                        }
                } else
                if (exercise.equals("jono")) {
                        System.out.println("Lisätään jonoon yksi kerrallaan");
                        Jono j = new Jono();
                        for (int i = 0; i < size; i++) {
```

```
                        Object o = (Object) (i + 1);
                        j.addFirst(o);
                        j.printContent();
                }
                System.out.println("Poistetaan jonosta yksi kerrallaan");
                for (int i = 0; i < size; i++) {
                        Object o = j.removeLast();
                        j.printContent();
                }
        } else {
                // init knots required for the graph shown in task 3 of the exercise 1
                Solmu a = new Solmu('A');
                Solmu b = new Solmu('B');
                Solmu c = new Solmu('C');
                Solmu d = new Solmu('D');
                Solmu e = new Solmu('E');
                Solmu f = new Solmu('F');

                // ..and then add neighbours for each knot as required
                Solmu[] bn = {a};
                b.setNeighbours(bn);
                Solmu[] an = {b, c, d};
                a.setNeighbours(an);
                Solmu[] cn= {a, f};
                c.setNeighbours(cn);
                Solmu[] dn = {a, f};
                d.setNeighbours(dn);
                Solmu[] en = {f};
                e.setNeighbours(en);
                Solmu[] fn = {e, c, d};
                f.setNeighbours(fn);

                // Look through the graph depth-wise..
                searchDepthWise(a, null);

                // clean knots for the next task..
                a.clean();
                b.clean();
                c.clean();
                d.clean();
                e.clean();
                f.clean();

                searchWidthWise(a, null);

                System.out.println("\nKokeile myös komento 'java AIExercises jono' ja 'java AI

        }
}

    static void searchDepthWise(Solmu start, Solmu target) {

            System.out.println("\nSyvyyssuuntainen haku");
```

```
                Pino solmulista = new Pino();
                solmulista.push(start);
                start.added = true;

                while (solmulista.getSize() > 0) {
                        System.out.print("Solmulista: ");
                        solmulista.printContent();

                        Solmu s = (Solmu) solmulista.pop().data;
                        // Look for target..
                        if (target != null && s.info.toString().equals(target.info.toString())) {
                                System.out.println("Ratkaisu");
                                return;
                        }


                        // Add neighbours, add same neighbour only once..
                        Solmu n;
                        int i = 0;
                        while (s.neighbours != null && s.neighbours.length > i) {
                                n = s.neighbours[i];
                                if (n.added == false) {
                                        solmulista.push(n);
                                        n.added = true;
                                }
                                i++;
                        }
                }
                System.out.println("Ei ratkaisua");
        }

        static void searchWidthWise(Solmu start, Solmu target) {

                System.out.println("\nLeveyssuuntainen haku");
                Jono solmulista = new Jono();
//                solmulista.addFirstSolmu(x);
                solmulista.addLastSolmu(start);
                start.added = true;

                while (solmulista.getSize() > 0) {
                        System.out.print("Solmulista: ");
                        solmulista.printContent();

//                        Solmu s = solmulista.removeLastSolmu();
                        Solmu s = solmulista.removeFirstSolmu();
                        // Look for target..
                        if (target != null && s.info.toString().equals(target.info.toString())) {
                                System.out.println("Ratkaisu");
                                return;
                        }

                        // Add neighbours, add same neighbour only once..
                        Solmu n;
```

```
                        int i = 0;
                        while (s.neighbours != null && s.neighbours.length > i) {
                                n = s.neighbours[i];
                                if (n.added == false) {
//                                        solmulista.addFirstSolmu(n);
                                        solmulista.addLastSolmu(n);
                                        n.added = true;
                                }
                                i++;
                        }
                }
                System.out.println("Ei ratkaisua");
        }
}
```

Koeajo:

```
$ java AIExercise1

Syvyyssuuntainen haku
Solmulista: A(B:C:D) |
Solmulista: D(A:F) | C(A:F) | B(A) |
Solmulista: F(E:C:D) | C(A:F) | B(A) |
Solmulista: E(F) | C(A:F) | B(A) |
Solmulista: C(A:F) | B(A) |
Solmulista: B(A) |
Ei ratkaisua

Leveyssuuntainen haku
Solmulista: [A](B:C:D)
Solmulista: [B](A) [C](A:F) [D](A:F)
Solmulista: [C](A:F) [D](A:F)
Solmulista: [D](A:F) [F](E:C:D)
Solmulista: [F](E:C:D)
Solmulista: [E](F)
Ei ratkaisua

Kokeile myös komento 'java AIExercises jono' ja 'java AIExercises pino'..
```