Artificial Intelligence
Örebro University,

DT2016, DT2022,

First Exam

Date: January, 12th 2015

Teacher: Franziska Klügl (Tel. 3925 and 070-6689179)

Allowed Aids: Calculator and/or Swedish/English Dictionary

Grading:

DT2016: 20 points are required for degree 3; 30 points for degree 4 and 35 points for degree 5.
DT2022: 20 points are required for degree G, 32.5 points required for degree VG.

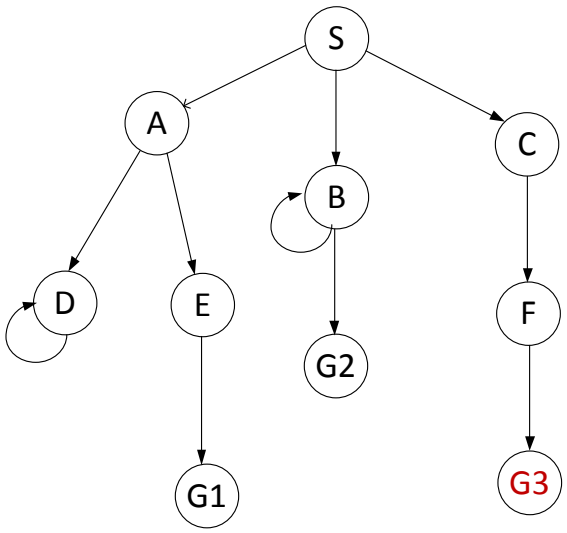
Remarks:

- You may answer in Swedish or English.
- Use a new sheet for the solution of each task.
- Read the task descriptions carefully and do not forget to answer **all** questions.
- Motivate and justify all your answers in sufficient detail.
  Never answer with just one word, but argue and explain. This helps the examinator to understand your reasoning and evaluate your work.

Artificial Intelligence Exam, HT 2014/2

Task 1: Search

In the following search graph (tree) there are three solutions G1, G2 and G3. Assume that the search terminates when the first goal state is reached.



In the exam there was G2 here. I looked carefully through the solution and always decided for the sake of the student if it was not clear what G2 the student meant.

a) **3 Points**

Search the tree starting from S using

- **Depth-First Search**
- **Iterative Deepening Search with Depth-Limited Depth First Search**

Precisely give the contents of the different lists involved in the search. The differences between Depth First Search and Iterative Deepening Search must be clearly shown. At which goal your search is ending?

- Depth-First Search: We just consider a frontier/agenda list and put the full path on the list, as done in the lecture:
  - [S] → Expanding S
  - [SA,SB,SC] → Expanding SA
  - [SAD, SAE, SB, SC] → Expanding SAD
  - [SADD, SAE, SB, SC] → Expanding SADD
  - [SADDD, SAE, SB, SC] … this leads to an endless loop – so no solution could be found.
- Alternative Solution: Depth-First Search with Explored List, only expanding paths with nodes that we did not yet consider:
  - [S] and [] → Expanding S
  - [SA,SB,SC] and [S] → Expanding SA
  - [SAD, SAE, SB, SC] and [S, A] → Expanding SAD – while doing so we put D on the explored list and see that SAD cannot be further expanded with a node that we did not yet handle.
  - So next step is to expand SAE to SAEG1

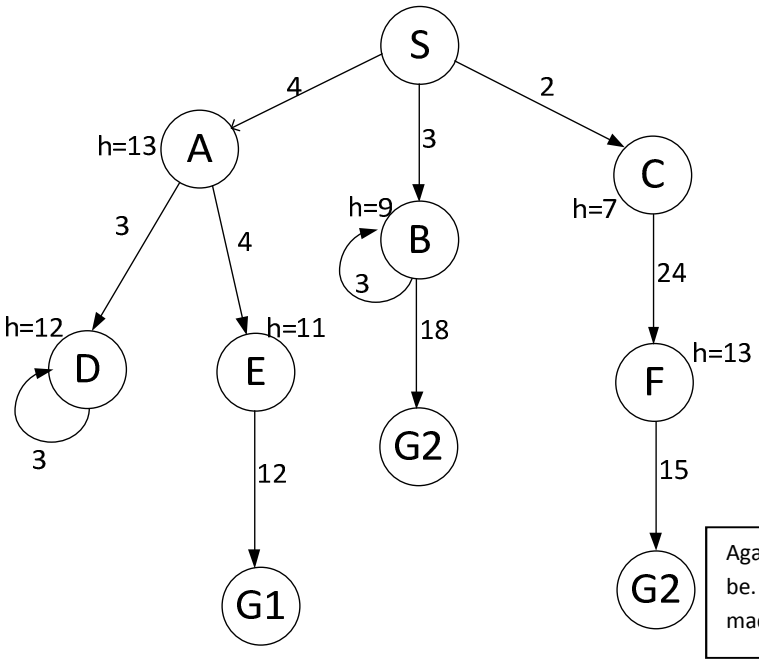
- [SAEG1, SB, SC] and [S, A, D] → Expanding SAEG1 – first is to test whether G1 is a Goal → it is, we are done.

- For iterative Deepending we do need to handle explored lists. The important aspects here were that a student shows clearly that a) it is a depth-first search (correct sequence of nodes) and b) that there is a limit in search which is increased:
  - Level 1: consider only [S]
  - Level 2:
    - [S]
    - Expands S to: [SA,SB,SC]
  - Level 3:
    - [S]
    - Expands S to [SA,SB,SC]
    - Expands SA → [SAD, SAE, SB, SC]
    - SAD and SAE are not expanded, but SB → [SBG2, SC]
    - Depending on when exactly we do the goal check, the search stops now or after or in the next iteration after SC is also expanded without success.
    - …

    At the end we find G2 as the goal.

b) 3 Points

Information about costs has been added to the search tree above: Numbers at links refer to costs from one node to the other; information at nodes ("h=…") are the values of the heuristic, that means the estimation of future costs between that node and the next goal.



Search this tree with information starting from S using

- **Best-First Search**
- **A***

Clearly describe what node you expand in which sequence and how the different involved lists are changing. The difference between Best First Search and A* must become clear. At which goal are the different search techniques ending?

Best-First Search:

- Expanding S: Agenda: [A (13), B (9), C(7)]; Explored = [S]
- C is the node with the smallest heuristic
  → Expanding C: Agenda [A (13), B(9) F(13)]; Explored [S,C]
- Expanding B as it is the currently best with respect to distance to goal
  → Expanding B: Agenda [A (13), F (13), G2(0)]; Explored [S, C, B]
- Expanding G2 with 0 distance to goal → done

A* - Agenda is the "frontier" and contains nodes with cost-so-far and heuristic

- Expanding S: Agenda [A (4+13), B(3+9), C(2+7)]; Explored [S (0+?)]
- Expanding C: Agenda [A (4+13), B(3+9), F(26+13)]; Explored [S (0+?), C(2+7)]
- Expanding B: Agenda [A (4+13), F(26+13), G2(21+0)]; Explored [S (0+?), C(2+7), B(3+9)]

- Expanding A: Agenda [D(7+12), E(8+11), F(26+13), G2(21+0)];
  Explored [S (0+?), B(3+9), A (4+13), C(2+7)]
- Expanding D or E? We start with D: D has only D as successor. D is already in Explored, the new way is 3 higher than before, D is not returning to the agenda (Costs from D to D)…
- Expanding E: [G1(20+0), F(26+13), G2(21+0)]; Explored [S (0+?), C(2+7), B(3+9), A (4+13), D(7+12)]
- Expanding G1 as it is the shortest overall …

For receiving the full amount of points, one needs to clearly show what reasoning is done for selecting the next node. In which format or how condensed is not important, if content is there.

c) **2 Points**

The standard Best First Search uses future costs as heuristic for guiding the search. Would it make sense to use the costs collected so far as heuristic to identify the next node to expand – e.g. by expanding the node with the lowest costs? Discuss this question.

A heuristic shall guide the search so that the nodes explored are the ones that are more probable to directly lead to the goal. The costs-so-far does not tell about the distance to the goal. It makes no sense.

One of the students gave a very good explanation: the student explained that it does not make sense by giving a mesh-network with the start somewhere in the center and the goal on one side… using that heuristic this would rather mean some form of breadth-first search.

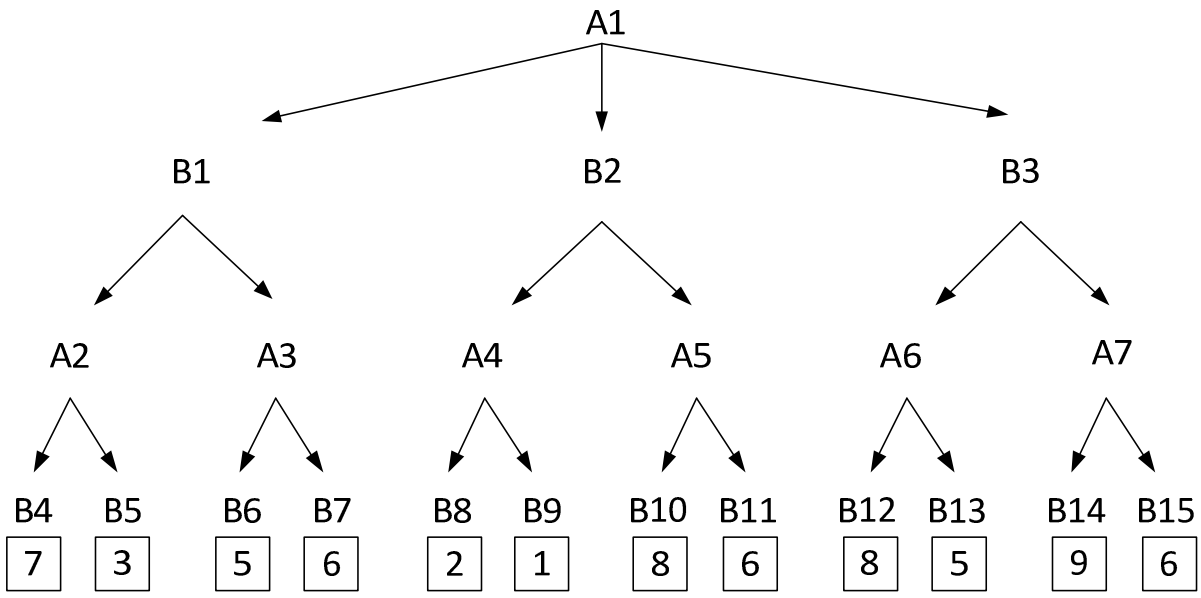
d) **2 Points**

How an iterative deepening version of A* could look like? Under which circumstances such an iterative A* could make sense?

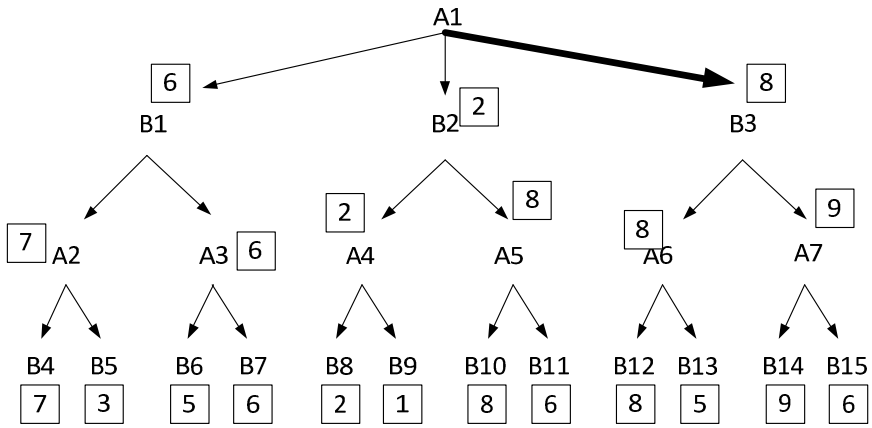
Yes, makes sense as one could save memory on omitting the explored list. The limit could be a max. overall-cost to which one searches.

Task 2: Game Tree Search

Consider the following game tree for a 2-player game. All nodes starting with A are nodes in which the MAX-player needs to take a decision; all nodes starting with B belong to the MIN-player. The leafs of the tree represent evaluations of how the game develops for the MAX-player in the respective branch.



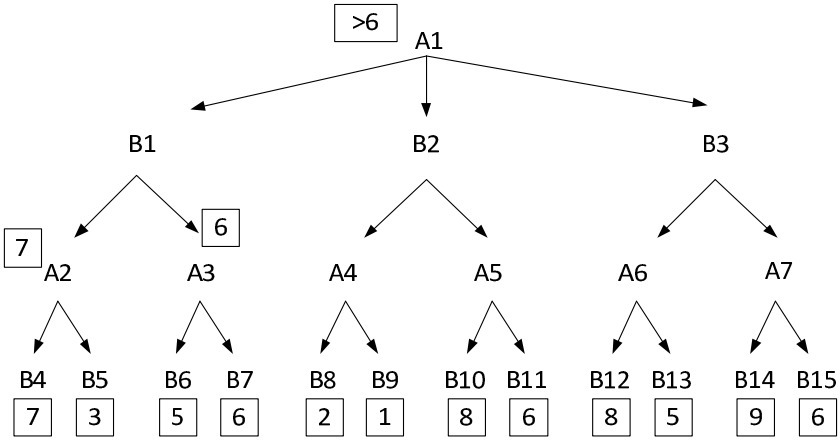
a) **3 points**
Which action the MAX-player shall take in the initial state A1? Use the MINIMAX algorithm for answering this question. Give all intermediate evaluation of game states.

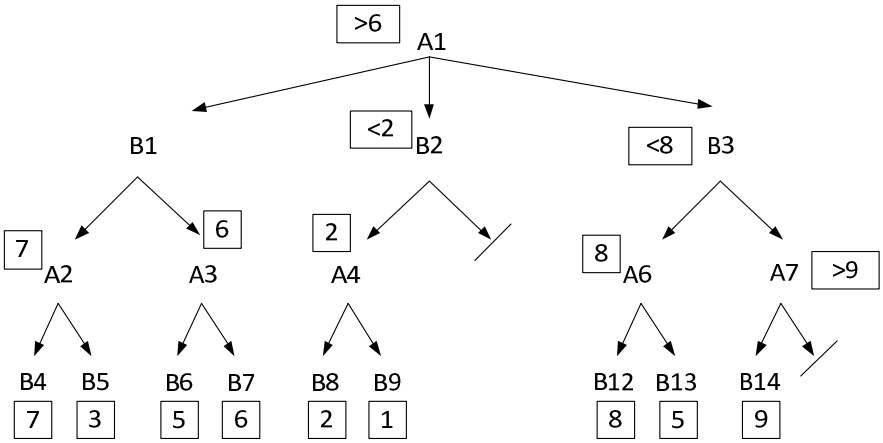


b) **3 points**

Use MINIMAX with alpha-beta-pruning on this tree. What branches do not need to be expanded? Explain why you did not consider these parts of the tree.

After expanding the first branch completely:





B3 becomes 8 and thus we know that the action leading to B3 is selected.

c) **1 points**

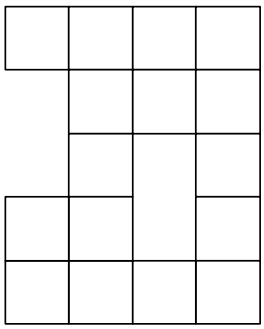
Would you say that alpha-beta pruning per se is a **heuristic** approach?

What does heuristic approach mean? Use additional (problem specific) information to guide search; Additional information is based on estimation, not something sure. In that sense, alpha-beta pruning has nothing to do with guessing and uncertain knowledge and just helps us to save some effort.

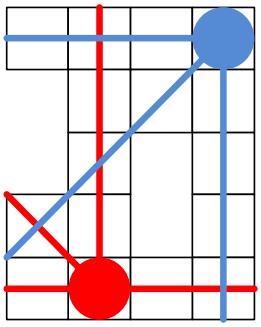
Task 3: Problem Modeling and Solving

5 points

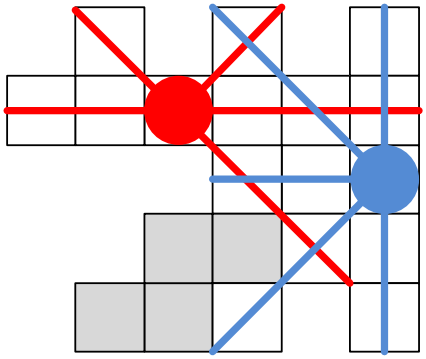
We have seen many mathematical puzzles as examples for AI problems and tried to solve them: Prominent examples that we had a (short) look at are Logical puzzles, N-queen problems, N-puzzles,...
A nice game that is available for smart phones is named “Octopus”: A number of octopuses needs to be placed on a (more or less complex) grid so that all cells are covered by the arms of the octopuses. Intersections are admissible. For example the following grid:



Can be solved by placing 2 octopuses:



For a grid like the following more than 2 octopuses would be necessary:



Which of the technique that are listed below would you use to determine the minimum number of octopuses and their placement for an arbitrary, given grid?

a) Uninformed State-Space Search (such as Depth-first, Breadth-first search)?
b) Heuristic Search (such as A* search)?
c) Local Search (such as Greedy/Hill-Climbing Search, Simulated Annealing or Genetic Algorithms)?
d) Constraint Satisfaction?
e) Automated Planning?

Select one of those techniques and argue why you would use it (and not the others). Also describe how you would model the problem in a way that this technique can be used.

Artificial Intelligence Exam, HT 2014/2

Actually any solution could give the full points. The important aspect is the argumentation: How you would use it and why you would use it and not the others. Clearly some of the techniques are more apt than others, nevertheless if clearly explained how to use, also solutions that I would not choose received full points.

a) Solving the problem with uninformed search → we need to formulate the problem as a search problem → we would start with a first empty grid and then the successor states would be all possible positions of one octopus and etc… and the successor states to those are all possible positions where to put the next one, etc… the branching factor makes that not a good choice, but it is possible.
b) Heuristic search would be not be much better, unless we would need to find a good heuristic. Maybe that could be based on the number of free cells that an octopus covers.
c) Local search would be a good alternative. One takes a solution using a number of randomly positioned octopus'. In addition to the state, one needs to define some mutation function – that means how to change a (partial) solution to another – hopefully better – one. An evaluation function could be the number of free cells. This evaluation function should be minimized with zero the absolute minimum. When we have this, we could use a Simulated Annealing.
d) Constraint satisfaction is actually not as a good option as one would suspect. The question then would be what are the variables and what are the values: a variable is a cell and the octopus number would be the value. Assigning a value would consequently a set of numbers to the corresponding cells. The problem here is the question how to formulate the constraints. Setting one octopus does not constrain the potential values of the others.
e) Planning makes not really sense here. One would need to define a planning domain description talking about what would be the effect of placing an octopus and what would be the prerequisite of placing. As there is no constraint, there is actually no condition that restricts not putting an octopus kernel on each cell, so what should be the prerequisite?

So, it is a clear optimization problem and thus a search problem. But, any rather complete argumentation, even ending up in constraint satisfaction received full points. There were reductions of points if the selection of technique did not fit to the description of the solution, also if only one solution is given without considering why not any other.

Task 4: Knowledge Representation

a) **3 Points**

Consider the following natural language sentences and two candidates for potential translations into predicate logic.

Give **for each** translation, whether it is a correct translation, an erroneous translation or simply wrong syntax.

Line2 and Line3 both stop at Studentgatan.

stopsAt(Line2 ∧ Line3, Studentgatan)

stopsAt (Line2, Studentgatan) ∧ stopsAt (Line3, Studentgatan)

first is wrong syntax, second is correct

There is a time in which neither Line 2 nor Line 3 are scheduled

∃t *time(t) ∧ ¬scheduled(Line2, t) ∧ ¬scheduled(Line3, t)*

∃t *time(t) ⇒ (¬scheduled(Line2, t) ∧ ¬scheduled(Line3, t))*

First is correct, second is erroneous as the statement "if t is a time" does not make sense, what if t is not a time?

Two drivers who drive busses on the same line, cannot have lunch together

∀*x,y (driver(x,Buss1) ∧ driver(y, Buss2) ∧* (*line*(*Buss1*)=*line*(Buss2))) ⇒ ¬*lunchTogether*(*x*,*y*)

∀*x,y* ¬ *driver*(*x, Buss1*) ∨ ¬ *driver*(*y, Buss2*) ∨ ¬ (*line*(*Buss1*)=*line*(Buss2)) ∨ ¬*lunchTogether*(*x*,*y*)

First is correct, second corresponds to the first one, is just CNF, so both are correct

b) Consider the following problem:

Victor has been murdered; Arthur, Bert and Carl are the only suspects – that means one of them is the murderer. They make the following statements:

Arthur says that Bert was the victim's friend, but Carl hated the victim

Bert says: He was out of town at the day of the murder and he did not even know Victor

Carl says: He saw Arthur and Bert with the victim just before the murder

We use the following predicates for formalizing those statements:

friend(x): x was a friend of Viktor

enemy(x): x was an enemy of Viktor

out(x): x was out of town during the murder

saw(x): x was seen just before the murder

Artificial Intelligence Exam, HT 2014/2

We want to find out who is the murderer. You shall do that using Resolution. The statements of the three suspects can be easily translated to (predicate) logic:

Arthur states	Bert states	Carl states
friend(Bert) enemy(Carl)	out(Bert) ¬knows(Bert)	saw(Bert) saw(Arthur)

Yet, for doing reasoning, the knowledge base must be completed by adding the following commonsense statements:

∀x: friend(x) ⇒ knows(x)
(If x is a friend of the victim, then he knows the victim)

∀x: enemy(x) ⇒ knows(x)
(If x is a friend of the victim, then he knows the victim)

∀x saw(x) ⇒ [¬out(x) ∧ knows(x)]
If x has been seen with the victim just before the murderer, x was not out of town and x knows the victim

a) **3 points**
Translate the three common sense knowledge into CNF (Conjunctive Normal Form)

friend(x) ⇒ knows(x)
enemy(y) ⇒ knows(y)
saw(z) ⇒ ¬out(z) ∧ knows(z)

in CNF:
¬friend(x)∨ knows(x)
¬enemy(y) ∨ knows(y)
¬saw(z) ∨ (¬out(z) ∧ knows(z)) ≡ [¬saw(z) ∨ ¬out(z)] ∧ [¬saw(z) ∨ knows(z)]

b) **5 points**
Use Resolution to find whether there is a contradiction when using all statements of each of the suspects for finding out who is a full liar. The statements of one produce the contradiction. This is the murderer. Who is the murderer?
Explain what you are doing in each step.

the knowledge base is
1 ¬friend(x)∨ knows(x)
2 ¬enemy(y) ∨ knows(y)

3 ¬saw(z) ∨ ¬out(z)
4 ¬saw(z) ∨ knows(z)
5(A) friend(Bert)
6(A) enemy(Carl)
7(B) out(Bert),
8(B) ¬knows(Bert)
9(C) saw(Bert),
10(C) saw(Arthur)

5(A)+1 → 11(A) knows(Bert)
11(A)+8(B)→ □

So, now we know that either A or B did lie as their statements are inconsistent.
Ok, what about A's second statement: enemy(Carl) – we cannot tell anything about that.
Ok, what about B's first statement (the second one was used in the reasoning above):

7(B)+3 → 12: ¬saw(Bert)
12+9(C) → □
So again, either Bert or Carl did lie…

➔ Bert is the murderer

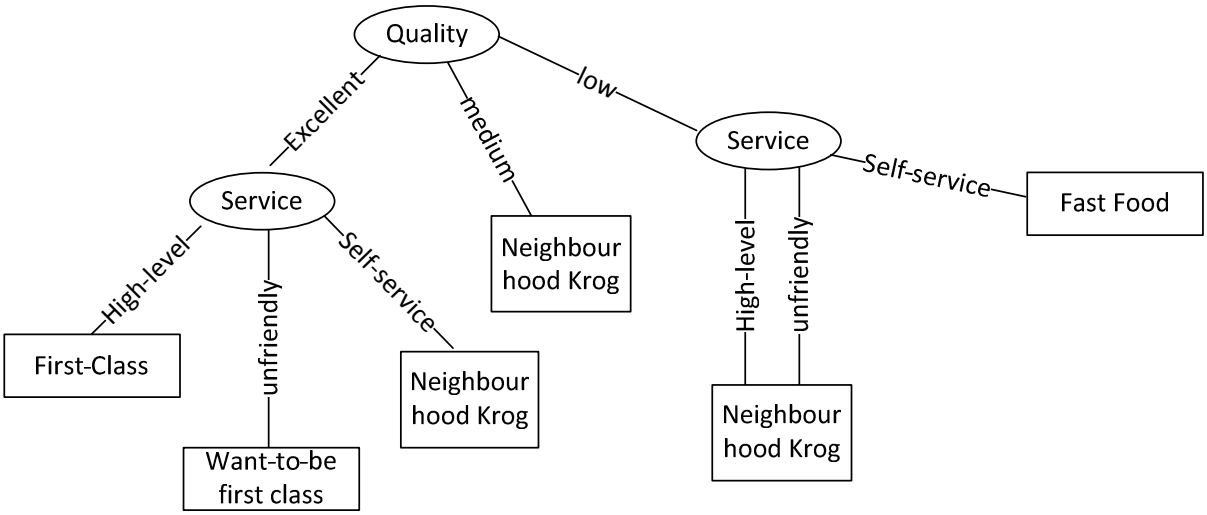
Task 4: Decision Trees

a.) **3 Points**

Sketch a decision tree for representing restaurants. Use attributes such as

- Open daily (boolean value)
- Size (large, medium, small)
- Sound (Noisy, Low-or-none)
- Service (high-level, unfriendly, self-service)
- Quality (Excellent, medium, low)

With a decision attribute representing a class of restaurants such as “First-class elite kitchen”, “neighbourhood restaurant”, “Fast-food restaurant”, etc.



Anything was accepted as long it was recognizable as decision tree. Does not need to include all attributes.

b.) **2 Points**

Explain the basic idea of the ID3 Algorithm for learning Decision Trees

ID3 learns a decision tree from a set of examples, each consisting of a set of attribute-value pairs. One determines which attribute to use as the root of the decision tree based on the information gain of the different attributes. One chooses the attribute that separates the list of examples best with respect to the decision attribute. This is recursively repeated for the different branches below the root and thus for subsets of examples that are represented by the branch.

c.) **2 Points**

Describe the basic idea behind Case-based Reasoning and explain why it is different from using a decision tree when applied to similar problems.

Both techniques are based on cases, often represented as a set of attribute-value pairs. A decision tree and case-based reasoning can be used to solve the same problem: finding a solution to a case. In a decision tree one would test one attribute after the other when going down the tree. So the Decision Tree also gives some form of question-strategy. Case-based reasoning on the other side uses a similarity measure to find the most similar case – it compares complete cases to the query-case. It is much more flexible and robust as a technique. If there is any slight inconsistency in the query, case based reasoning still can give reasonable results. Going down a decision tree has no backtracking, it simply goes down and that's it.