Проект по функционално програмиране-практикум
„Игра на животни“
Василен М. Чижов, 3 курс, Информатика, ФМИ

1. Анализ на задачата

Задачата реално се свежда до отделяне на елемент от множество по дадени критерии. В случая елемента, който искаме да отделим е животното, което потребителят си е намислил. Идеята е чрез краен брой стъпки от множеството на животните да отделим това животно което потребителят си е намислил. На разположение освен множеството от животни, имаме и множество от въпроси свързани с животните, на които може да се отговаря само с да или не. Също така, първоначално знаем, че отговорите на някои от въпросите за някои от животните са ни известни (не е задължително да знаем отговора на всеки въпрос за всяко животно първоначално). Начина по който получаваме информация за намисленото животно е чрез задаване на въпроси към потребителя, като той е задължен да отговаря правилно.

Двата най-очевидни варианта за описание на тази задача според мен са: или като пазим за всеки въпрос множества от животните, за които той отговаря с да/не/не е сигурно, или ако за всяко животно пазим вектор с толкова на брой компоненти коклкото въпроси имаме засега, като на всеки въпрос съответства дадена позиция във вектора, и всяка позиция може да приема само 3 стойности съответстващи на отговорите да/не/не е сигурно. За простота съм избрал първия вариант.

Нека A е множеството от всички животни, които знаем засега. Нека за всеки въпрос qi (i=1,..,n – нека имаме n въпроса първоначално) имаме три множества Ai, Bi, Ki. Нека Ai се състои от всички животни за които знаем, че отговарят с да на qi. Нека Bi се състои от всички животни за които знаем, че отговарят с не на qi. И нека Ki се състои от останалите животни (т.е. тези за които не знаем какъв е отговора на въпроса qi). Ясно е, че AiUBiUKi = A, т.е. не е нужно да пазим и четирите множества A, Ai, Bi, Ki, тък като всяко едно може да се изрази чрез другите 3, но за удобство ще приемаме, че имаме всички множества. Тогава ще дефинираме какъв ще е алгоритъма за намаляване на началното можество A. Нека C0 = A, D0 = $\varnothing$, Q0 = ((q1,A1,B1,K1), …, (qn,An,Bn,Kn)) са работните ни множества на стъпка 0 (Q0 е списък). Нека на стъпка k>=0, работните множества са Ck, Dk и Qk. Тогава получаваме Ck+1, Dk+1 и Qk+1 по

следния начин:
Нека Qk = ((qi1,Ai1,Bi1,Ki1), …, (qis,Ais,Bis,Kis)), тогава Q'k+1 = ((qi1, Ck∩Ai1, Ck∩Bi1, DkUKi1), …, (qis, Ck∩Ais, Ck∩Bis, DkUKis)). Ако подредим елементите на Q'k+1 в нарастващ ред по модула от 4-тата им компонента първо (за елемент j ще е |DkUKij|) и абсолютната стойност от разликата на модулите на 2-та и 3-тата им компонента (за елемнт j ще е ||Aij|-|Bij||) второ. Ще получим списъка Q"k+1, ако сега от този списък махаме подред елементи от началото докато не стигнем до елемнт q = (q, qA, qB, qK) : (|qA|,|qB|)!=(0,0) и |qA|!=|Ck| и |qB|!=|Ck|, или докато не остане само един елемент, ще получим Q'"k+1. Нека зададем въпрос q на потребителя, ако отговора е да Ck+1 = qA, а ако е не Ck+1 = qB; Dk+1 = qK. Qk+1 се получава от Q'"k+1 като махнем q (т.е. като махнем първия елемент). Ясно е че след краен брой стъпки Qk ще е празен на някоя стъпка или Ck ще е празен.

Ако Ck се окаже празен, очевидно не сме познали животното, или поради това че просто то не участва в множеството с животните, или защото за някой въпрос не сме имали достатъчно информация (или защото потребителят лъже). Ако животното не е било от множеството с животните трябва да го добавим в него и да го добавим във множеството за несигурния отговор на всеки въпрос. Независимо дали животното е било от множеството с животните, можем да използваме зададените въпроси и получените отговори за да „подобрим“ можеството от въпросите.

Ако Qk се окаже празен, ясно е че не можем да задаваме повече въпроси. Можем да питаме потребителя дали животното е първият елемент на Ck, ако е - всичко е ОК. Ако не е, ще разберем какво е било намисленото животно. Ако намисленото животно е от Ck, просто трябва да научим въпроси, с които да го различаваме от всяко друго животно от Ck. Ако животното не е от Ck, и не е от множеството с животните, трябва да го добавим в множеството с животните, да „подобрим“ множеството от въпросите чрез зададените въпроси и получените отговори, и да се научим да го различаваме от другите животни в Ck. Ако животното не е от Ck, и е от множеството с животните, тогава не сме имали достатъчно информация за някой въпрос (или потребителя е излъгал), трябва да „подобрим“ множеството от въпросите чрез зададените въпроси и получените отговори, и да се научим да го различаваме от другите животни в Ck. Това са всичките възможни случаи за край.

Нека сега разгледаме защо списъка в стъпката се сортира по

модула от 4-тата на компонента на елементите първо. Идеята на това е да се задават с приоритет въпроси с минимална несигурност (т.е. тяхното Ki множество да е максимално малко), защото ако зададем въпрос за който |Ki| е голямо вероятността е по-голяма да „отхвърлим“ намисленото животно при отговора (защото има по-голяма вероятност то да е в Ki), а от там нататък няма как да го върнем в Ck. Причината вторият признак на сортиране да е по абсолютната стойност от разликата на модулите на 2-та и 3-тата компонента е (това сортиране не е задължително!) за да може всяка игра да не продължава „прекалено дълго“ или „кратко“. За пример нека изберем въпрос за който |Ai| = 1, |Bi|=|A|-1. Наистина с този въпрос можем да „познаем“ животното на първия ход, но вероятността е 1/|A|. Тогава ако подбираме просто въпросите по втори признак минимума на модула на някое от множествата им Ai или Bi, ще имаме по-дълги игри, и по-къси които да компенсират за това (вероятността в края на краищата се събира до 1), ако обаче от друга страна сортираме по ||Ai|-|Bi||, ще избираме въпроси където разликата между големината на двете множества е минимална – тогава игрите би трябвало да завършват за някаква „средна“ продължителност.

Нека сега разгледаме въпроса когато потребителят „лъже“, това лесно може да се провери при достигане до края на играта. Ако той е „излъгал“ на някой от въпросите за които имаме правилна информация и животното е от познатите, тогава ако просто съспоставим зададените въпроси, получените отговори и съответстващите им множества можем да намерим точно на кой въпрос е „излъгал“ потребителят.

2. Описание на основните функции и структури от данни изпозвани при решаването на зададения проблем

Една структура е използвана за поддържането на множеството от животните и въпросите (както и множествата на всеки въпрос) в паметта. Структурата има следния вид: (animals_set (q1 A1 B1 K1) … (qn An Bn Kn)). Ако искаме да получим само множеството от животните това се постига лесно чрез car а ако искаме множеството от въпроси cdr. animals_set е множество от низове (имената на животните) подредени лексикографски.

Главните функции за работа с animals_set са: (known-animal? a G) проверява дали животно а е част от G, където G е наредено множество като animals_set; (add-animal-l a l) добавя животно а в множеството l, ако го няма в l; (remove-animal-l a l) премахва животно а от множеството l, ако го има в l; (add-animal a G) добавя животното а както в множеството на животните така и във всяко множество Ki от структурата G.

Главните функции за работа с множеството от въпросите са: (known-question? q G) проверява дали въпроса q присъства в множеството от въпросите G; (add-question q G) добавя въпроса q в структурата G; (question-info q G) нека q==qi от множеството на въпросите G, тогава функцията връща (qi Ai Bi Ki); (associate-animal-question a answer q G) нека qi е от G и q==qi тогава ако answer==#t премества а от Ki в Ai иначе в Bi.

Това бяха основните функции за работа с множеството на животните, множеството на въпросите и структурата от двете. Стъпката разгледана в 1 е реализирана в (step A K G), където A е Ck, K е Dk, а G = Qk. (half-step G T) изпълнява сечението и обединението на можествата от Qk (съответства на G) чрез (transform-question A K q) за всеки въпрос, като след това ги вмъква сортирани по обсъдените критерии от 1 в T един по един чрез (insert-sorted q T). Функцията (choose T) изпълнява премахването на елементи от Q'"k+1 обсъдено в 1.

Играта започва със (start-game G tryhard) което реално започва в 0-та стъпка и продължава натам, докато не стигне до някой от

обсъдените случаи от 1. Проверката дали животно е от Ck става чрез (known-animal-l a l), 'подобрението“ на въпросите става чрез (update-path a path G), което взима записаните подред досега въпроси с отговорите им от path и за всеки от тях се опитва да подобри множеството от въпроси в G. “Подобряването“ на множеството от въпроси така, че да може да различава намислентото животно от тези в Ck става чрез (update-difference l G). Проверката дали потребителят е излъгал става чрез (did-you-lie a path G) която връща списък с въпросите на които е излъгал и отговора който е трябвало да даде.

Структурата с множество от животните и въпросите може да бъде записана на файл чрез (write-db filename G), а такава структура може да бъде прочетена от файл чрез (read-db filename).

Функцията (main T tryhard) е функцията която трябва да бъде викана от потребителя, тя принципно вътрешно използва (start-game T tryhard) като просто пита дали потребителя дали иска да продължи след край на играта, дали иска да запише новата получена структура и т.н.

3. Идеи за бъдещи подобрения

- Повечето операции свързани с множеството на животните и множеството на въпросите могат да използват бинарно търсене.
- Ефикасността (процесорно време/памет) може да се увеличи чрез ползването на мутиращи операции.
- Може предишни игри да се записват и по тях да се отсъди за по-ефективна поредица от въпроси които да бъдат задавани.
- Може да се добави възможността програмата да проверява дали е даде грешен въпрос като отговор.