Project Problems

SNU 4910.210, Fall 2014

Chung-Kil Hur

due: 12/21(Sun), 24:00

Problem 1 (**10%**) "먼지폭풍"

지구에서 보낸 화성표면 탐사로봇은 2032년 현재 100개 이상이고, 그 갯수가 빠르게 증가하고있다. 지구에 없는 귀중한 금속자원이 화성표면에서 속속 발견되고 있기 때문이다.

화성 궤도에는 모선이 돌고 있어서, 화성표면에서 임무를 수행하는 수많은 로봇들과 수시로 데이타와 명령을 주고 받도록 되어있다.

이 모선이 하는 일 중 하나는, 화성대기에 먼지 폭풍이 예상되면 표면 탐사로봇들을 재빨리 대피소로 대피시키는 것이다. 모선에서는 화성표면의 로봇들에게 대피해야 할 대피소를 지정해주고 대피 명령을 내리는 것이다.

하나의 대피소에는 하나의 로봇만 수용할 수 있고, 대피 명령을 받은 로봇은 직

선경로로 대피소를 향해 전속력으로 이동하게 되어있다.

문제는 로봇이 대피소로 이동중에 다른 로봇과 충돌할 수 있다는 점이다. 충돌할 일이 없도록 각각의 로봇들에게 대피소를 할당해 줘야 한다.

이 일을 하는 프로그램 shelterAssign을 아래의 모듈 타입에 맞게 작성하라.

```
module type DUSTSTORM =
 sig
  type robot = string                    (* robot's name *)
  type shelter = int                     (* shelta's id number *)
  type location = int * int              (* 화성에서의 위도와 경도 *)
  type robot_locs = (robot * location) list
  type shelter_locs = (shelter * location) list
  val shelterAssign: robot_locs -> shelter_locs -> (robot * shelter) list
 end
```

화성은 2차원 평면이라고하자. 원구나 도넛모양으로 끝들이 연결되지 않는다. 위도와 경도는 왼쪽 위를 (0,0)으로 오른쪽 아래는 (100,100)으로 한다.

그리고, 왜 이 프로그램이 항상 올바른 답을 유한시간내에 내는 지를 프로그램에 코멘트로 설명하라.

- "올바른 답": 로봇과 대피소의 짝이 주어지면 절대 로봇끼리 직선경로 이동중에 충돌의 가능성이 없어야 한다.
- "유한시간내": 항상 끝나야 한다.
- 힌트: 처음에는 무작위로 로봇과 대피소를 짝짓는다. 그리고 다음을 반복한다. 직선이동 중 충돌할 수 있는 짝을 찾아 $\cdots$ 하기.

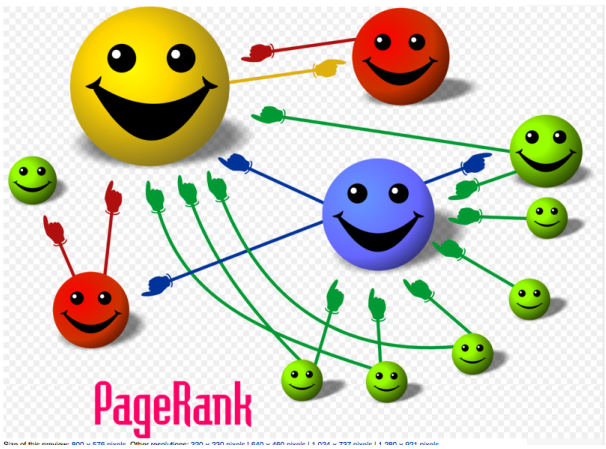
□

Problem 2 (**10%**) "랭킹추측"

구글의 페이지랭크(PageRank) 기술은 전세계 모든 웹페이지들의 "중요도" 순위를 예측하는 기술이다. 구글 서치엔진은 사용자가 입력한 문구를 가진 웹페이지들을 추린후, 이 기술을 이용해서 웹페이지들을 "중요도" 순서대로 보여주려는 것이다. 이 중요도 순서가 사용자가 읽고 싶은 페이지 순서와 일치하길 바라면서.

그럼 어떻게 이 "중요도"를 계산해야할까? 구글 창업자인 Page와 Brin의 아이디어는 이렇다:

- 가정: 사람들이 자주 방문하는 페이지가 "중요한" 페이지이다.
- 그렇다면, 페이지를 방문하는 빈도를 알아내면 될 텐데, 어떻게 알아낼 것인가?



- 아이디어: 불특정 다수의 사람이 인터넷 서핑을 하는 과정을 모사(simulation)해서 예측해 보자. 사람들은 한 페이지에서 시작해서 링크를 따라 이런저런 페이지를 방문할 것이고, 종종 새로운 페이지에서부터 다시 서핑을 시작하곤 할 것이다. 이런 경우를 모사해서 각 페이지가 방문되는 상대적인 비율을 계산하면, 그 비율이 그 페이지의 "중요도"가 아닐까.
- 그렇다면, 어떻게 그런 시뮬레이션을 할 것인가?
- 답: 사람들이 방문하는 과정을 *Markov chain*으로 모델링하면 안성맞춤이다.

Example 1 *Markov chain*으로 모델링하는 예.

다음과 같은 경우를 생각해 보자. 서울시와 세종시 사이에는 매년 인구이동이 있다. 서울에서는 매년 인구의 5%가 세종시로 이주하고, 반대로 세종시에서는 인구의 15%가 서울시로 이주한다고 하자. n-번째 해의 서울시의 인구 S_n과 세종시의 인구 J_n은 다음의 관계를 만족할 것이다:

$$\begin{aligned} S_n &= 0.95 \times S_{n-1} + 0.15 \times J_{n-1} \\ J_n &= 0.05 \times S_{n-1} + 0.85 \times J_{n-1} \end{aligned}$$

즉,

$$\begin{pmatrix} S_n \\ J_n \end{pmatrix} = \begin{pmatrix} 0.95 & 0.15 \\ 0.05 & 0.85 \end{pmatrix} \begin{pmatrix} S_{n-1} \\ J_{n-1} \end{pmatrix}$$

그러면, 초기 인구를 각각 S_0, J_0으로 놓고, 위의 점화식을 연쇄적으로 계산해 가면 서울시와 세종시의 인구가 최종적으로 어떻게 될 지를 알수있을 것이다.

위의 행렬과 같이, 각 렬(column)의 합이 모두 1인 행렬을 Markov행렬이라고 하는데, Markov행렬로 표현되는 점화식이 연쇄적으로 만드는 값들을 *Markov chain*이라고 한다. □

이제 다시 페이지랭킹 문제로 돌아오면, 수많은 사람들이 웹페이지를 방문하면서 형성되는 각 페이지별 방문비율은 다음과 같이 *Markov chain*으로 모델링 될 수 있을 것이다:

- 전세계 웹 페이지가 3개있고 A, B, C라고 하자. 각 페이지에는 다른 페이지로 향하는 링크가 매달려 있을 것이고, 사람들은 이 링크를 따라 방문을 한다. 매번 페이지 A가 방문될 비율은 A, B, C에서 A로 가는 링크의 갯수에 비례한다고 하자. 예를 들어, 페이지 B에 링크가 3개인데, 그 중 하나가 A로 오는 것이면 B를 방문한 사용자는 1/3의 비율로 A를 방문한다고 할 수 있다. 다른 페이지들에 대해서도 서로 이런 관계로 방문비율의 변화를 표현할 수 있을 것이다. 예를들어 다음과 같이: (페이지마다 링크의 목적지는 각각 다르다고 가정하고, 페이지 A에는 링크가 두 개 있고, B에는 세 개, C에는 하나가 있다고 가정):

$$\begin{aligned} A_n &= \tfrac{1}{2} \times A_{n-1} + \tfrac{1}{3} \times B_{n-1} + \tfrac{0}{1} \times C_{n-1} \\ B_n &= \tfrac{1}{2} \times A_{n-1} + \tfrac{1}{3} \times B_{n-1} + \tfrac{1}{1} \times C_{n-1} \\ C_n &= \tfrac{0}{2} \times A_{n-1} + \tfrac{1}{3} \times B_{n-1} + \tfrac{0}{1} \times C_{n-1} \end{aligned}$$

  즉,

$$\begin{pmatrix} A_n \\ B_n \\ C_n \end{pmatrix} = \begin{pmatrix} 1/2 & 1/3 & 0 \\ 1/2 & 1/3 & 1 \\ 0 & 1/3 & 0 \end{pmatrix} \begin{pmatrix} A_{n-1} \\ B_{n-1} \\ C_{n-1} \end{pmatrix}$$

- 일반적으로 총 웹페이지수가 N개라면 다음과 같은 $N \times N$ Markov 행렬 $\mathcal{M}$이 만들어진다.
  - $x_{ij} \in \mathcal{M}$: i-페이지가 j-페이지를 링크하면 $1/r_i$ ($r_i = i$-페이지에 있는 링크 수) 아니면 0.
  - 그러면, 시간이 지나면서 페이지들이 방문되는 비율의 변화는 다음의 점화식으로 표현된다:

$$S_n = \mathcal{M} S_{n-1}$$

– 목표: 궁극적으로 어떻게 될 지를 계산하고 싶다. 즉, 극한 S

$$S = \lim_{i \to \infty} S_i$$

를 계산하려는 것이다. 초기 벡터 S_0는 모든 엔트리를 $1/N$으로 하면 될 것이다(모든 페이지가 방문을 시작하는 페이지일 비율이 동일하다는 가정).

– 문제: 항상 그런 *Markov chain*의 극한이 유일하게 존재하는가? 그래서 프로그램으로 단순하게 연쇄과정을 반복하는 작업을 짜면 항상 그 프로그램은 유한시간내에 끝나게 될까?

– 힌트: Perron-Frobenius 정리

위의 답을 찾고, 주어진 행렬이 Markov 행렬이고 그 극한이 유일하게 존재하도록 변환시킨 후 그 극한을 계산하는 함수 `markov_limit`를 작성하라. 아래의 모듈타입을 만족하도록한다. (함수들 `row`, `column`, `add_row`, `add_column`, `size`, `ij`등은 매트릭스를 만들고 사용하는 함수들이다.)

```
module type MARKOV =
 sig
    type matrix
    val row: float list -> matrix
    val column: float list -> matrix
    val add_row: float list -> matrix -> matrix
    val add_column: float list -> matrix -> matrix
    val size: matrix -> int * int    (* numbers of columns and rows *)
    val ij: matrix -> int -> int -> float
    (*
      Given a Markov matrix and an initial column,
      markov_limit returns the limit of the Markov chain.
    *)
    val markov_limit: matrix -> matrix -> matrix
 end
```

□

Problem 3 (**10%**) "트랜스포머"

한 세계의 물건을 다른 세계의 물건으로 변환시키는 "트랜스포머"를 만들어보자. 우리 변환의 대상물과 결과물은 정수를 다루는 프로그램들이다.

- 변환 대상은 다음의 규칙으로 만들어지는 정수식 프로그램이다:

E	$\rightarrow$	n	정수
	|	E+E	덧셈식
	|	-E	부호변경식
	|	`read`	정수입력식
	|	`if` $E\ E\ E$	조건식
	|	`repeat` $E\ E$	반복식

입력식 `read`는 외부에서 입력받은 정수를 뜻한다. 입력은 -100과 100사이의 정수라고 가정한다. 조건식

$$\texttt{if}\ E_1\ E_2\ E_3$$

의 결과는 E_1의 값에 따라 결정된다: E_1의 값이 0이 아니면 E_2의 값이되고 0이면 E_3의 값이 된다. 반복식

$$\texttt{repeat}\ E_1\ E_2$$

은 E_1의 값이 음수가 아닐때만 정의되는 데, 결과는 E_2의 값을 E_1의 값만큼 반복해서 더한 결과이다. 즉, (E_1의 값) $\times$ (E_2의 값)이다. 따라서 E_1이 0이면 결과는 0이다.

- 변환 결과물은 다음의 규칙으로 만들어지는 명령문 프로그램이다:

C	$\rightarrow$	x `has` n	변수 x가 정수 n을 가진다
	$\mid$	x `has` x	다른 변수의 값을 가진다
	$\mid$	x `has` x`+`x	다른 변수의 덧셈결과를 가진다
	$\mid$	x `has` x`-`x	다른 변수의 뺄셈결과를 가진다
	$\mid$	x `has read`	입력된 정수를 가진다
	$\mid$	`say` x	변수를 출력한다
	$\mid$	`goto` ℓ `on` x	변수에 따라 점프
	$\mid$	ℓ`:` C	태그가 붙은 명령문
	$\mid$	C `;` C	순서대로 실행되는 명령문들

보다시피 변수가 중심이다. 모든 값을 항상 변수에 넣고 변수를 가지고 계산이 진행된다. 따라서 변수를 사용전에는 항상 그 변수에 어떤 값이 들어가 있어야 한다.

명령문

x `has read`

는 외부에서 정수입력을 받아 변수 x에 넣는다. 입력은 -100과 100사이의 정수라고 가정한다. 명령문

`goto` ℓ `on` x

은 변수 x의 값이 0이면 일없이 마치고, 0이 아니면 태그 ℓ이 붙은 명령문으로 점프한다.

예를들어 다음의 프로그램은 외부에 1을 출력한다:

```
    x has 1 ;
    y has 2 ;
    x has x+y ;
    goto L on x ;
    x has 0 ;
L:  z has x-y ;
    say z
```

다음의 프로그램은 실행될 수 없는 프로그램이다. 변수 z가 값을 가지지

도 않았는데 사용되기 때문이다:

```
x has 1 ;
y has x+x ;
z has y+z ;
say z
```

- 변환기는 변환대상 정수식과 "같은 일을 하는" 명령문 프로그램을 내놓는다. 이 명령문 프로그램은 원래 정수식이 계산하는 값과 같은 값을 출력해야한다.

- 변환전 확인사항: 변환해서는 안되는 "의미없는" 정수식이 있다. 변환기는 의미없는 정수식이 입력으로 들어오면 변환을 거부해야 한다.

  의미없는 정수식은 한가지 경우밖에 없다. 반복식

  $$\texttt{repeat}\ E_1\ E_2$$

  에서 E_1의 값이 음수인 경우다. 이런 정수식은 그 계산이 정의되어있지 않다.

  변환기는 번역대상물인 정수식에 이런 의미없는 반복식이 있는지 확인해야 한다. 번역하기전에 E_1의 값을 예측해서 음수여부를 체크해야 한다.

- 변환후 확인사항: 변환결과로 만들어 지는 C프로그램이 제대로 동작하는 지를 확인해야 한다.

  이 확인은 궁극적으로 변환대상과 변환결과가 같은 일을 하는 지를 체크해야 하지만, 여기서는 다음의 두가지 경우를 확인하도록 한다.

  - 변수가 사용되기 전에 항상 값이 넣어졌는지.
  - 점프하는 대상 명령문이 제대로 정의되어있는지. 즉, 점프할 태그가 붙은 명령문이 딱 하나 있는지.

변환기 `transform`과 검사기들 `check_exp`와 `check_cmd`는 아래의 모듈 타입을 만족하도록 한다:

```
module type TRANS =
 sig
  type exp = Num of int
           | Add of exp * exp
```

```
          | Minus of exp
          | Read
          | If of exp * exp * exp
          | Repeat of exp * exp
 type var = string
 type tag = string
 type cmd = HasNum of var * int
          | HasVar of var * var
          | HasSum of var * var * var
          | HasSub of var * var * var
          | HasRead of var
          | Say of var
          | Goto of tag * var
          | Tag of tag * cmd
          | Seq of cmd * cmd
 val transform: exp -> cmd
 val check_exp: exp -> bool
 val check_cmd: cmd -> bool
end
```

□