Kapitel 1

Lösungsidee

Im Prinzip ist die Paderboxaufgabe ziemlich simpel, nur das Erzeugen der zufälligen Zustände mitsamt ihren Eigenschaften ist ein bisschen trickreich.
Beim Verteilen der Lampen tritt das Problem auf, sicherzustellen, dass jede Lampe wirklich mit exakt zwei Zuständen verbunden ist, und nicht etwa zweimal mit demselben oder mit mehr als zwei. Ebenso muss jeder Zustand mit zwei unterschiedlichen Lampen verbunden sein.
Es gibt daher für einen Zustand nur diese 3 möglichen Lampenkonfigurationen: $\{0,1\}$, $\{0,2\}$ und $\{1,2\}$.
Interessanterweise erfüllen diese Konfigurationen, wenn jede genau einmal auftritt, die an die Lampen gestellte Bedingung, dass jede Lampe mit zwei Zuständen verbunden sein muss. Umgekehrt gilt, dass es, wenn eine dieser Konfigurationen doppelt vorkommt, für den letzten Zustand nur eine Lampe gibt, was zur ungültigen Konfiguration $\{a,b\}$ führt.
Man kann also einfach diese drei Konfigurationen zufällig mischen (echtes Mischen, nicht zu verwechseln mit dem Mischen bei MergeSort) und dann der Reihe nach auf die Zustände aufteilen, um alle gültige, zufällige Lampenzustandzuordnung finden zu können.

Beim Erzeugen der Zustandsübergänge entsteht nun ein ähnliches Problem, nämlich das der unerreichbaren Zustände. Werden die Verbindungen zwischen zwei Zuständen nämlich wirklich rein zufällig gewählt, kann es leicht passieren, dass ein oder sogar zwei Zustände unerreichbar sind.

Ob solche Situationen verhindert werden sollen, ist nicht ganz klar. Einerseits verhält sich dann die Paderbox wie eine Box mit weniger als drei Zuständen, was es ein einfacher macht, die Zustände zu ermitteln; anderenseits ist es sicherlich interessanter das Spiel zu spielen, wenn man ab und zu durch einfachere Boxen überrascht wird. Daher werden zuerst die Zustandsübergänge rein zufällig ermittelt und danach, je nach Wunsch des Benutzers, alle noch nicht erreichbaren Zustände, zufällig mit Verbindungen versehen.
Wenn nur ein Zustand nicht erreichbar ist, kann man entweder vom Startzustand oder von dem erreichbaren Zustand aus eine Verbindung zu diesem unerreichbaren schaffen. Welchen man davon nimmt kann man einen (Pseudo-)zufallszahlengenerator entscheiden lassen, allerdings muss man beim Startzustand aufpassen, nicht die Verbindung zum bisher bereits erreichbaren zu kappen, wenn man eine Verbindung durch die neue ersetzt. Man muss also die Verbindung ersetzten die nicht zu einem anderen Zustand führt, es sei denn der unerreichbare Zustand hat eine Verbindung (von ihm) zu dem erreichbaren oder der Startzustand hat zwei Verbindungen zum bisher erreichbaren.
Sind zwei Zustände unerreichbar, also der Startzustand hat nur Verbindungen zu sich selbst, kann man eine davon durch eine Verbindung zu einem der anderen Zustände ersetzen und hat dann entweder den Fall, dass nur ein Zustand erreichbar ist, oder das sogar beide schon erreichbar sind (letzteres, falls der neue Zustand eine Verbindung zum anderen besitzt).

Die letzten beiden Forderungen, zufällige Konfiguration der Lampen und des Startzustands sind relativ trivial. Da es keinerlei Einschränkungen gibt, welche Lampen beim Start eingeschaltet sein dürfen, kann man einfach drei zufällige Wahrheitswerte erzeugen und entsprechend die Lampen schalten.
Den Startzustand kann man auch einfach zufällig wählen, vorausgesetzt man überprüft anschließend wie oben beschrieben die Erreichbarkeit der anderen Zustände. Allerdings gibt es von außen keinerlei Möglichkeit die Zustände von einander zu unterscheiden, außer deren Verknüpfungen mit den Lampen. Da aber diese Verknüpfungen bereits zufällig sind, ist es sinnlos noch einen zufälligen Startzustand zu wählen.

Bei der Ausgabe der Lösung könnte man nun, die Zustände auflisten, ihre Zuordnung zu den Lampen und in welche Zustände die Box beim Drücken der Tasten in diesem Zustand wechselt. Da aber die Zustände von außen sowieso nicht zu erkennen sind, kann es dann mehrere Lösungen geben, die

dadurch entstehen, dass man die Zustände entsprechend ändert.
Deshalb orientiert sich meine Paderbox an den Lampen und listet auf, welche Lampen geändert werden, wenn im Schritt vorher bestimmte andere Lampen geändert wurden. Damit man den Startzustand erkennt, wird dieser zuerst angezeigt.

Kapitel 2

Programmdokumentation

Beim Starten des Programms wird zuerst die Prozedure **init** aufgerufen, die eine zufällige Ausgangssituation erzeugt, die dann von der Prozedure **reset** geladen und angezeigt wird.
Dann beginnt der Hauptteil des Programms, in dem in einer Endlosschleife die gedrückte Taste eingelesen wird und dann darauf reagiert wird. Diese Tasten mitsamt Verhalten sind:

0, 1	Ruft **change** auf, um den Zustand zu wechseln.
r	Ruft **reset** auf, um zum Anfangszustand zurückzukehren.
s	Ruft **printSolution** auf, um die Lösung anzuzeigen.
q	Beendet das Program und zeigt vorher die Lösung an.

Nun folgt die Beschreibung dieser Funktionen:
Die Prozedure **i**nitialisiert zuerst zwei temporäre Hilfsarrays mit Namen **i** `visitable`, das die erreichbaren Zustände angibt und `freeLightConfig`, das die Nummern aller bisher noch nicht zugewiesen Lampenpaare speichert.
Anschließend werden die Zustände erzeugt, indem immer zufällige Lampenzuordnung aus `freeLightConfig` gewählt wird und Verknüpfungen zu beliebigen anderen Zuständen geraten werden. Für den Zustand i, wird die Lampenzuordnung in `state[i].lightsID` und die Verknüpfungen in dem zwei elementigen Array `state[i].changes` gespeichert.
Anschließend

Kapitel 3

Ablaufprotokoll

Kapitel 4

Quellcode

```
program paderbox;
{$mode objfpc}
uses crt;
const CHANGE_LIGHTS:array[0..2,0..1] of longint=
          ((0,1),(0,2),(1,2)); //lights to change in

var states: array[0..2] of record //Liste der Zustände
      changes:array[0..1] of longint; //Übergänge
      lightsID: longint; //damit verbundene Lampen
    end;
    //(start) light-status
    slights,lights: array[0..2] of boolean;
    //current state
    state:longint;

  //————print the status of the lights————
  procedure printStatus();
  const LIGHTSTATE:array[boolean] of char = ('O','X');
  var i:longint;
  begin
    for i:=0 to 2 do
      write(LIGHTSTATE[lights[i]]);
    write(' ');
  end;

  //————print the programming of the box————
  procedure printSolution();
```

```
const LIGHT_NAMES: array[0..2] of string=('++_','+_+','_++');
var i:integer;
begin
  writeln();
  writeln('Printing solution:');
  for i:=0 to 2 do
    writeln('  ',LIGHT_NAMES[states[i].lightsID],': ',
            ' 0->',LIGHT_NAMES[states[states[i].changes[0]].lightsID],
            ' 1->',LIGHT_NAMES[states[states[i].changes[1]].lightsID]);
   writeln();
end;

//————-creates a random programming————
procedure init();
  //creates a connection to new, from old or state 0
  procedure addLink(new, old: longint);
  begin
    if (random(1000)<500) then begin
      //link from start state
      if (states[new].changes[0]=old)or(states[new].changes[1]=old) or
         ((states[0].changes[0]=old)and(states[0].changes[1]=old)) then
          states[0].changes[random(2)]:=new
      else if states[0].changes[0]=old then states[0].changes[1]:=new
      else states[0].changes[0]:=new;
    end else //link from old
      states[old].changes[random(2)]:=new;
  end;

var i,j,k:integer;
    freeLightConfig: array[0..2] of longint;
    visitable: array[0..2] of boolean;
begin
  randomize;
  fillchar(visitable,sizeof(visitable),0);
  visitable[0]:=true;
  for i:=0 to 2 do
    freeLightConfig[i]:=i;
  //init states
  for i:=0 to 2 do begin
    //get light pair id
    j:=random(3-i);
    //save light id
    states[i].lightsID:=freeLightConfig[j];
    //remove light pair from freeLightConfig
    freeLightConfig[j]:=freeLightConfig[2-i];
```

```
    //creates random links
    for j:=0 to 1 do begin
      states[i].changes[j]:=random(3);
      if visitable[i] then //if j can be visited, the new, too
        visitable[states[i].changes[j]]:=true;
    end;
  end;
  //check states which cann't be visited
  if paramstr(1)<>'--allow-not-visitable-states' then begin
    //both are not visitable
    if (not visitable[1]) and (not visitable[2]) then begin
      //link to one
      j:=random(2);
      states[0].changes[j]:=random(2)+1;
      //update visit status
      visitable[states[0].changes[j]]:=true;
      visitable[states[states[0].changes[j]].changes[0]]:=true;
      visitable[states[states[0].changes[j]].changes[0]]:=true;
    end;
    //if one is not visitable call addLink
    if not visitable[1] then addLink(1,2)
    else if not visitable[2] then addLink(2,1);
  end;
  //random lights
  for i:=0 to 2 do
    slights[i]:=random(1000)<500;
end;

//————-change to start status————–
procedure reset();
begin
  state:=0;
  lights:=slights;
  writeln('state: 0');
  printStatus();
end;

//change status
procedure change(number: longint);
var i:integer;
begin
 with states[states[state].changes[number]] do
   for i:=0 to 1 do
     lights[CHANGE_LIGHTS[lightsId][i]]:=not lights[CHANGE_LIGHTS[lightsId][i]];
```

```
   printStatus();
  end;

//————main program————
var key:char;
    i:integer;
begin
  writeln('========Die Paderbox========');
  init();
  reset();

  while true do begin
    key:=ReadKey;
    writeln(key);
    case lowercase(key) of
      '0','1': change(ord(key)-ord('0'));
      'r': reset();
      's': begin
        printSolution();
        printStatus();
      end;
      'q': begin //quit
        printSolution();
        exit;
      end;
    end;
  end;
end.
```