alex-hw1-html.rmd

Question 1: Building a DAG (5 points)

1.1

Write out the factorization of the joint distribution implied by the DAG using mathematical notation. (1 point)

$P(A, E, S, O, R, T) == P(A)P(S)P(E|A, S)P(O|E)P(R|E)P(T|O, R)$

1.2

Rewrite the above factorization in *bnlearn*'s string representation. (1 point)

```
dagStr <- "[A][S][E|A:S][O|E][R|E][T|O:R]"
```

1.3

Use this to create a DAG in *bnlearn*. (1 point)

```
dag <- model2network(dagStr)
```

1.4

Print the class of the DAG object. (1 point)

```
class(dag)
```

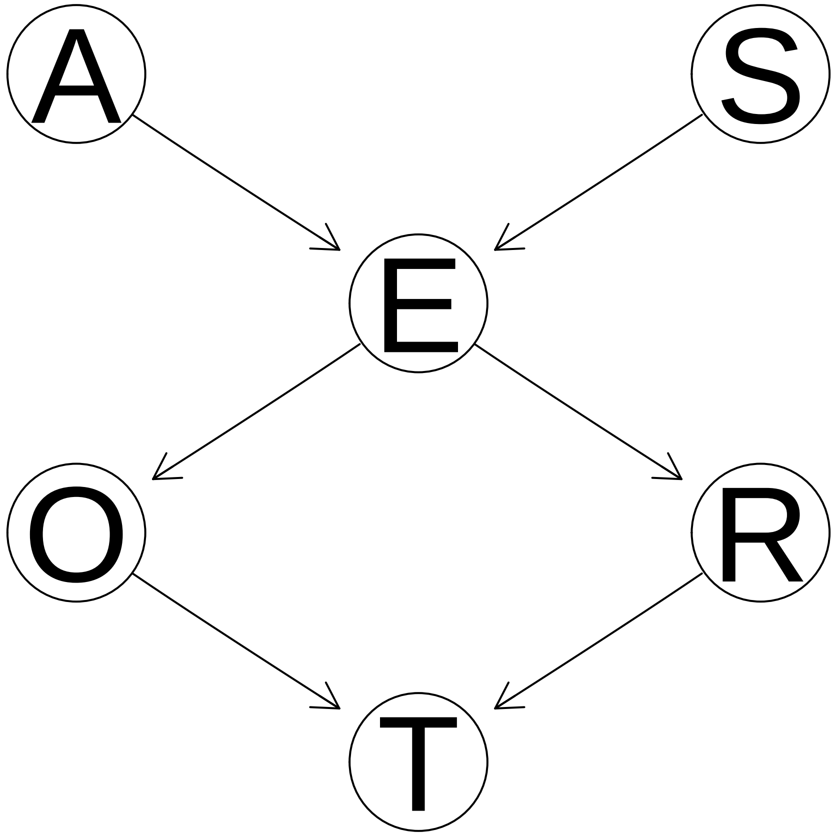
```
## [1] "bn"
```

1.5

Use `graphviz.plot` to plot the DAG. (1 point)

```
graphviz.plot(dag)
```

```
## Loading required namespace: Rgraphviz
```



Question 2: Experimenting with graph utilities (5 points)

2.1

Extract and print the nodes and arcs of the DAG you created in previous questions. (1 point)

```
nodes(dag)
```

```
## [1] "A" "E" "O" "R" "S" "T"
```

```
arcs(dag)
```

```
##      from to
## [1,] "A"  "E"
## [2,] "S"  "E"
## [3,] "E"  "O"
## [4,] "E"  "R"
## [5,] "O"  "T"
## [6,] "R"  "T"
```

2.2

Extract and print the parents and the children of each node using `parents` and `children` functions. (1 point)

```r
for(n in nodes(dag)) {
  cat(n, "'s parents are: '", parents(dag,n), "'.  ")
  cat(n, "'s children are: '", children(dag,n), "'")
  cat("\n")
}
```

```
## A 's parents are: '  '.  A 's children are: ' E '
## E 's parents are: ' A S '.  E 's children are: ' O R '
## O 's parents are: ' E '.  O 's children are: ' T '
## R 's parents are: ' E '.  R 's children are: ' T '
## S 's parents are: '  '.  S 's children are: ' E '
## T 's parents are: ' O R '.  T 's children are: '  '
```

2.3

Use the `mb` function to extract the Markov blanket of A, E, and T. (1 point)

```r
mb(dag, "A")
```

```
## [1] "E" "S"
```

```r
mb(dag, "E")
```

```
## [1] "A" "O" "R" "S"
```

```r
mb(dag, "T")
```

```
## [1] "O" "R"
```

2.4

How do you identify the Markov blanket from the DAG? (1 point) For node N Markov blanked is identified as its parents, its children and parents of those children, or in R code: (?) Should it be expressed in code?

2.5

Describe, in terms of coniditional independence (NOT in terms of the DAG) the definition of a Markov blanket. (1 point)

If M = Markov Blanket of variable Y from set of random variables S then Y when conditioned on M is indpendent on any subset X of set S provided $X \bigcap M = \emptyset$

Question 3: Conditional probability distribution (CPD) parameter estimation (5 points)

Bayesian network = DAG + CPD with specified parameters

3.1

Fit the parameters of the DAG from the data stored in survey2.txt using Bayesian estimation, and save the result into an object of class bn.fit. (2 points)

```
survey <- read.table("/Users/alex/i/causalML/HW/hw1_release/survey2.txt", header = TRUE)
survey[] <- lapply(survey, function(x) as.factor(x))
bn.bayesDefault <- bn.fit(dag, data = survey, method = "bayes")
```

3.2

Play with the Bayesian prior parameter **iss** and report the changes in the parameters learned from Bayesian network. Explain the changes. (3 points)

```
sink("bn.bayes_iss_default")
bn.fit(dag, data = survey, method = "bayes")
```

```
##
##   Bayesian network parameters
##
##   Parameters of node A (multinomial distribution)
##
## Conditional probability table:
##      adult       old     young
## 0.3575391 0.1578417 0.4846193
##
##   Parameters of node E (multinomial distribution)
##
## Conditional probability table:
##
## , , S = F
##
##       A
## E          adult       old     young
##   high 0.6389365 0.8446809 0.1558105
##   uni  0.3610635 0.1553191 0.8441895
##
## , , S = M
##
##       A
## E          adult       old     young
##   high 0.7191617 0.8913043 0.8099825
##   uni  0.2808383 0.1086957 0.1900175
##
##
##   Parameters of node O (multinomial distribution)
##
## Conditional probability table:
##
##       E
## O            high        uni
##   emp  0.98016416 0.96531303
```

```
##   self 0.01983584 0.03468697
##
##   Parameters of node R (multinomial distribution)
##
## Conditional probability table:
##
##        E
## R             high        uni
##   big   0.71751026 0.93824027
##   small 0.28248974 0.06175973
##
##   Parameters of node S (multinomial distribution)
##
## Conditional probability table:
##          F         M
## 0.5468986 0.4531014
##
##   Parameters of node T (multinomial distribution)
##
## Conditional probability table:
##
## , , R = big
##
##        O
## T              emp       self
##   car   0.71084719 0.68553459
##   other 0.13887569 0.15723270
##   train 0.15027712 0.15723270
##
## , , R = small
##
##        O
## T              emp       self
##   car   0.54655295 0.72549020
##   other 0.07746979 0.25490196
##   train 0.37597726 0.01960784
```

```r
sink()

sink("bn.bayes_iss_1")
bn.fit(dag, data = survey, method = "bayes", iss=1)
```

```
##
##   Bayesian network parameters
##
##   Parameters of node A (multinomial distribution)
##
## Conditional probability table:
##      adult       old     young
## 0.3575391 0.1578417 0.4846193
##
##   Parameters of node E (multinomial distribution)
##
## Conditional probability table:
```

```
##
## , , S = F
##
##       A
## E          adult       old     young
##   high 0.6389365 0.8446809 0.1558105
##   uni  0.3610635 0.1553191 0.8441895
##
## , , S = M
##
##       A
## E          adult       old     young
##   high 0.7191617 0.8913043 0.8099825
##   uni  0.2808383 0.1086957 0.1900175
##
##
##   Parameters of node O (multinomial distribution)
##
## Conditional probability table:
##
##       E
## O            high        uni
##   emp  0.98016416 0.96531303
##   self 0.01983584 0.03468697
##
##   Parameters of node R (multinomial distribution)
##
## Conditional probability table:
##
##        E
## R             high        uni
##   big   0.71751026 0.93824027
##   small 0.28248974 0.06175973
##
##   Parameters of node S (multinomial distribution)
##
## Conditional probability table:
##         F         M
## 0.5468986 0.4531014
##
##   Parameters of node T (multinomial distribution)
##
## Conditional probability table:
##
## , , R = big
##
##        O
## T              emp       self
##   car   0.71084719 0.68553459
##   other 0.13887569 0.15723270
##   train 0.15027712 0.15723270
##
## , , R = small
##
```

```
##        O
## T               emp       self
##   car   0.54655295 0.72549020
##   other 0.07746979 0.25490196
##   train 0.37597726 0.01960784
```

```r
sink()

sink("bn.bayes_iss_5")
bn.fit(dag, data = survey, method = "bayes", iss=5)
```

```
##
##   Bayesian network parameters
##
##   Parameters of node A (multinomial distribution)
##
## Conditional probability table:
##      adult       old     young
## 0.3573935 0.1588972 0.4837093
##
##   Parameters of node E (multinomial distribution)
##
## Conditional probability table:
##
## , , S = F
##
##       A
## E          adult       old     young
##   high 0.6379898 0.8389121 0.1568266
##   uni  0.3620102 0.1610879 0.8431734
##
## , , S = M
##
##       A
## E          adult       old     young
##   high 0.7181168 0.8873418 0.8078261
##   uni  0.2818832 0.1126582 0.1921739
##
##
##   Parameters of node O (multinomial distribution)
##
## Conditional probability table:
##
##       E
## O            high        uni
##   emp  0.97755102 0.96218487
##   self 0.02244898 0.03781513
##
##   Parameters of node R (multinomial distribution)
##
## Conditional probability table:
##
##        E
## R             high        uni
```

```
##   big   0.71632653 0.93529412
##   small 0.28367347 0.06470588
##
##   Parameters of node S (multinomial distribution)
##
## Conditional probability table:
##         F         M
## 0.5466165 0.4533835
##
##   Parameters of node T (multinomial distribution)
##
## Conditional probability table:
##
## , , R = big
##
##        O
## T              emp       self
##   car   0.71013118 0.66081871
##   other 0.13924451 0.16959064
##   train 0.15062431 0.16959064
##
## , , R = small
##
##        O
## T              emp       self
##   car   0.54474982 0.65079365
##   other 0.07963354 0.26984127
##   train 0.37561663 0.07936508
```

```r
sink()

sink("bn.bayes_iss_10")
bn.fit(dag, data = survey, method = "bayes", iss=10)
```

```
##
##   Bayesian network parameters
##
##   Parameters of node A (multinomial distribution)
##
## Conditional probability table:
##     adult       old     young
## 0.3572139 0.1601990 0.4825871
##
##   Parameters of node E (multinomial distribution)
##
## Conditional probability table:
##
## , , S = F
##
##       A
## E          adult       old     young
##   high 0.6368243 0.8319672 0.1580882
##   uni  0.3631757 0.1680328 0.8419118
##
```

```
## , , S = M
##
##       A
## E          adult       old     young
##   high 0.7168246 0.8825000 0.8051724
##   uni  0.2831754 0.1175000 0.1948276
##
##
##   Parameters of node O (multinomial distribution)
##
## Conditional probability table:
##
##       E
## O            high        uni
##   emp  0.97432432 0.95833333
##   self 0.02567568 0.04166667
##
##   Parameters of node R (multinomial distribution)
##
## Conditional probability table:
##
##        E
## R             high        uni
##   big   0.71486486 0.93166667
##   small 0.28513514 0.06833333
##
##   Parameters of node S (multinomial distribution)
##
## Conditional probability table:
##          F         M
## 0.5462687 0.4537313
##
##   Parameters of node T (multinomial distribution)
##
## Conditional probability table:
##
## , , R = big
##
##        O
## T               emp       self
##   car   0.70923999 0.63440860
##   other 0.13970356 0.18279570
##   train 0.15105645 0.18279570
##
## , , R = small
##
##        O
## T               emp       self
##   car   0.54253835 0.58974359
##   other 0.08228731 0.28205128
##   train 0.37517434 0.12820513
```

```r
sink()
```

```r
sink("bn.bayes_iss_100")
bn.fit(dag, data = survey, method = "bayes", iss=100)
```

```
##
##   Bayesian network parameters
##
##   Parameters of node A (multinomial distribution)
##
## Conditional probability table:
##      adult       old     young
## 0.3543860 0.1807018 0.4649123
##
##   Parameters of node E (multinomial distribution)
##
## Conditional probability table:
##
## , , S = F
##
##       A
## E          adult       old     young
##   high 0.6187683 0.7425150 0.1793103
##   uni  0.3812317 0.2574850 0.8206897
##
## , , S = M
##
##       A
## E          adult       old     young
##   high 0.6959315 0.8122449 0.7641791
##   uni  0.3040685 0.1877551 0.2358209
##
##
##   Parameters of node O (multinomial distribution)
##
## Conditional probability table:
##
##       E
## O            high        uni
##   emp  0.92289157 0.89855072
##   self 0.07710843 0.10144928
##
##   Parameters of node R (multinomial distribution)
##
## Conditional probability table:
##
##        E
## R            high       uni
##   big   0.6915663 0.8753623
##   small 0.3084337 0.1246377
##
##   Parameters of node S (multinomial distribution)
##
## Conditional probability table:
##         F         M
```

```
## 0.5407895 0.4592105
##
##   Parameters of node T (multinomial distribution)
##
## Conditional probability table:
##
## , , R = big
##
##        O
## T              emp      self
##   car   0.6938899 0.4561404
##   other 0.1476104 0.2719298
##   train 0.1584997 0.2719298
##
## , , R = small
##
##        O
## T              emp      self
##   car   0.5093897 0.3908046
##   other 0.1220657 0.3218391
##   train 0.3685446 0.2873563
```

```r
sink()

sink("bn.bayes_iss_1000")
bn.fit(dag, data = survey, method = "bayes", iss=1000)
```

```
##
##   Bayesian network parameters
##
##   Parameters of node A (multinomial distribution)
##
## Conditional probability table:
##      adult       old     young
## 0.3429719 0.2634538 0.3935743
##
##   Parameters of node E (multinomial distribution)
##
## Conditional probability table:
##
## , , S = F
##
##       A
## E          adult       old     young
##   high 0.5512010 0.5656402 0.3021277
##   uni  0.4487990 0.4343598 0.6978723
##
## , , S = M
##
##       A
## E          adult       old     young
##   high 0.5997819 0.6100719 0.6127389
##   uni  0.4002181 0.3899281 0.3872611
##
```

```
##
##   Parameters of node O (multinomial distribution)
##
## Conditional probability table:
##
##       E
## O           high       uni
##   emp  0.7028902 0.6729560
##   self 0.2971098 0.3270440
##
##   Parameters of node R (multinomial distribution)
##
## Conditional probability table:
##
##        E
## R            high       uni
##   big   0.5919075 0.6628931
##   small 0.4080925 0.3371069
##
##   Parameters of node S (multinomial distribution)
##
## Conditional probability table:
##         F         M
## 0.5186747 0.4813253
##
##   Parameters of node T (multinomial distribution)
##
## Conditional probability table:
##
## , , R = big
##
##        O
## T             emp      self
##   car   0.5893471 0.3510773
##   other 0.2014605 0.3244613
##   train 0.2091924 0.3244613
##
## , , R = small
##
##        O
## T             emp      self
##   car   0.4014532 0.3398950
##   other 0.2515895 0.3320210
##   train 0.3469573 0.3280840
```

```r
sink()

sink("bn.bayes_iss_1000000")
bn.fit(dag, data = survey, method = "bayes", iss=1000000)
```

```
##
##   Bayesian network parameters
##
##   Parameters of node A (multinomial distribution)
```

```
##
## Conditional probability table:
##      adult       old     young
## 0.3333493 0.3332174 0.3334333
##
##   Parameters of node E (multinomial distribution)
##
## Conditional probability table:
##
## , , S = F
##
##       A
## E          adult       old     young
##   high 0.5000810 0.5000810 0.4995356
##   uni  0.4999190 0.4999190 0.5004644
##
## , , S = M
##
##       A
## E          adult       old     young
##   high 0.5001828 0.5001529 0.5001769
##   uni  0.4998172 0.4998471 0.4998231
##
##
##   Parameters of node O (multinomial distribution)
##
## Conditional probability table:
##
##       E
## O           high       uni
##   emp  0.5003507 0.5002748
##   self 0.4996493 0.4997252
##
##   Parameters of node R (multinomial distribution)
##
## Conditional probability table:
##
##        E
## R            high       uni
##   big   0.5001589 0.5002588
##   small 0.4998411 0.4997412
##
##   Parameters of node S (multinomial distribution)
##
## Conditional probability table:
##        F        M
## 0.500031 0.499969
##
##   Parameters of node T (multinomial distribution)
##
## Conditional probability table:
##
## , , R = big
##
```

```
##        O
## T              emp      self
##   car   0.3341263 0.3333520
##   other 0.3329249 0.3333240
##   train 0.3329488 0.3333240
##
## , , R = small
##
##        O
## T              emp      self
##   car   0.3334333 0.3333400
##   other 0.3332134 0.3333320
##   train 0.3333533 0.3333280
```

```r
sink()
```

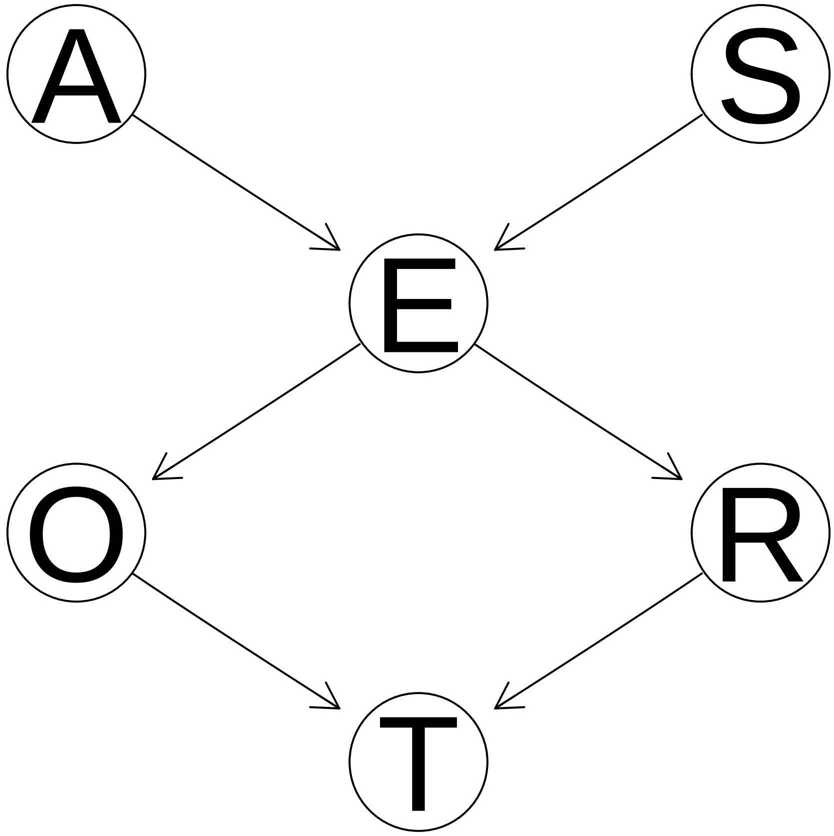
Explanation of differences in conditional propabilities for various `iss` argument:

Since `iss` represents sample size of imaginary prior distribution, which is uniform distribution, the large value for `iss` we use the closer conditional probabilities become to uniform distribution. It is especially demonstrated by latest example (iss=1000000), where calculated probabilities are very very close to uniform distribution. Also, when `iss` <= 10, calculated conditionl probabilities are almost identical, which means prior distribution does not play any significant role and since assigning prior distribution to uniform is an actually a very wild guess, it means smaller `iss` values are is what should be used in order to get "right" values of conditional probabilities (i.e. where effect of inital prior is eliminated)

5.1

Compute and plot the PDAG of the DAG for the survey data using the `cpdag` function. Call this PDAG P1 and the original DAG D1. How does P1 and D1 compare? Explain any similarities or differences. (1 point)

```r
d1 <- dag
p1 <- cpdag(d1)
graphviz.plot(p1)
```

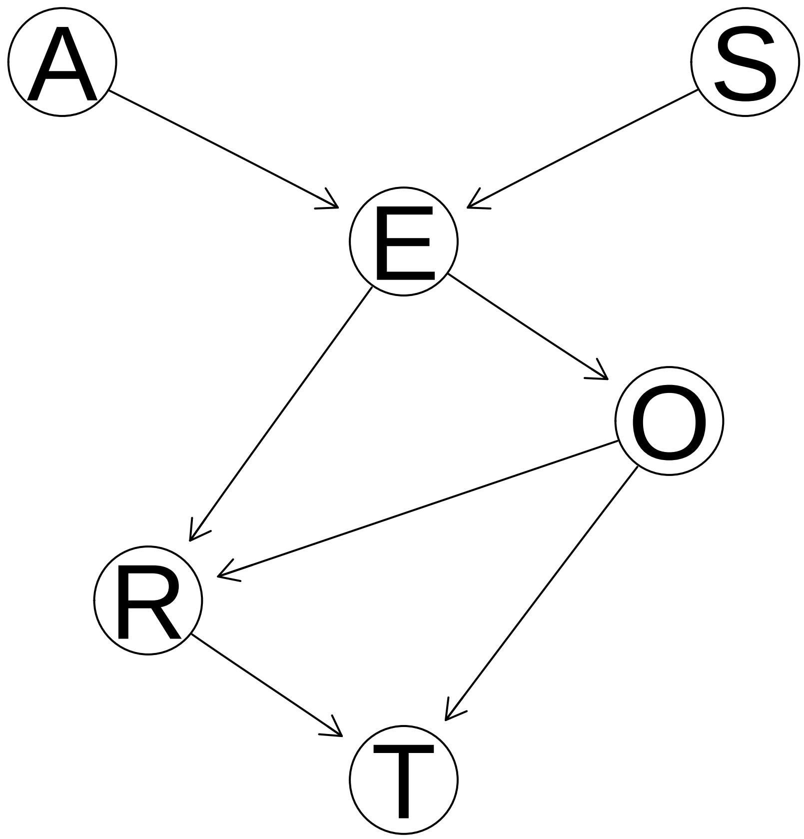


Explanation of result: p1 is identical to d1 because any arrow reversal in p1 would change set of open V-structures: i.e. reversing O->E would create new open V structure: O->E<-A which cannot be found in d1

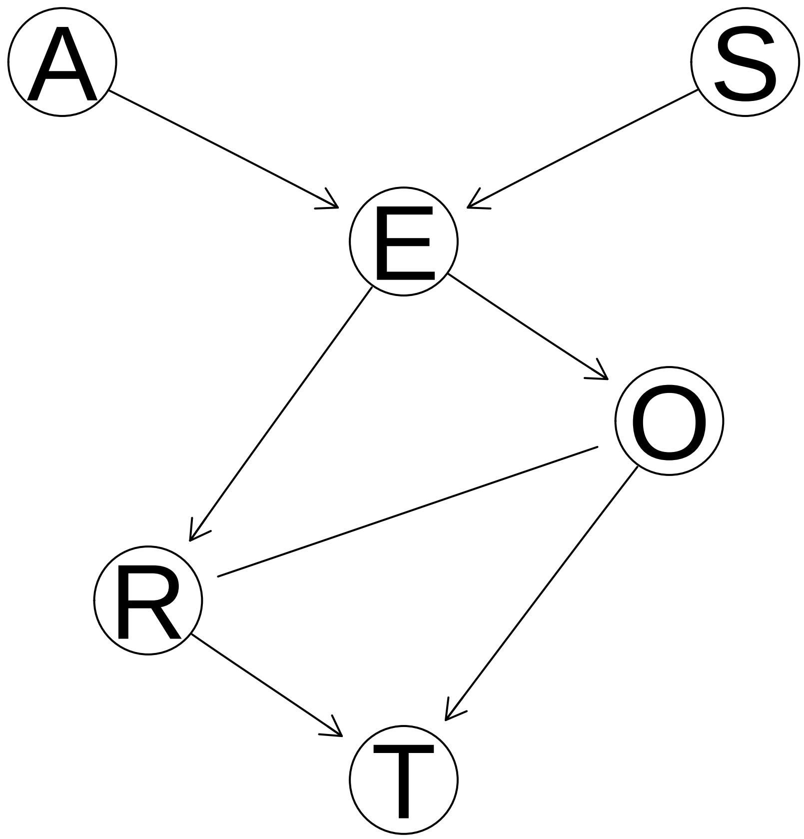
5.2 Create a DAG D2 that is the same as D1 except that it has a new arc from Occupation to Residence. This makes sense because surely somebody's

job determines where they live (or is it the other way around?). Note that this is a fine example of applying domain knowledge about the data generative process in causal model development. Plot the result with `graphviz.plot`. Now recompute a PDAG P2 from D2. What, if anything, is different between P1 and P2 and what explains these differences or lack of differences? (1 point)

```
d2 <- model2network("[A][S][E|A:S][O|E][R|E:O][T|O:R]")
graphviz.plot(d2)
```




```r
p2 <- cpdag(d2)
graphviz.plot(p2)
```



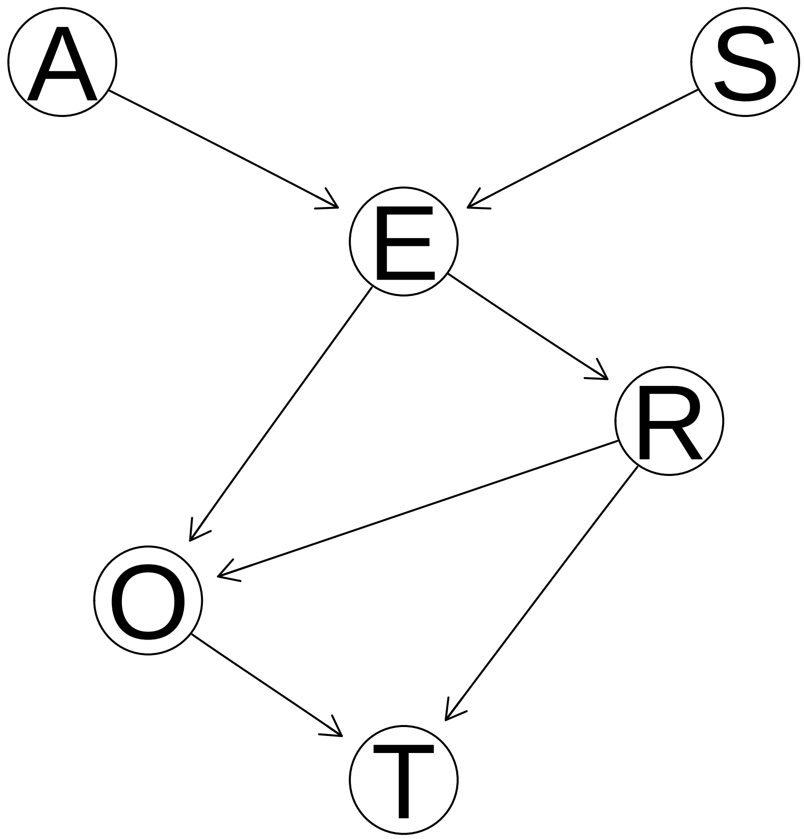
p1 is different than p2 and differences are:

- skeletal difference, p2 has an extra edge R - O
- Edge R-O has no direction, which meas whatever direction is assigned to it, set of open V-structures remain the same.

5.3 Create a third DAG D3 that is different from the second DAG

(with the O->R edge) but is in the same Markov equivalence class. Do this by reasoning about P2 – in other words look at P2 and create another DAG D3, such that `cpdag(D3)` will also produce P2. Plot D3. (1 point)

```
d3 <- model2network("[A][S][E|A:S][O|E:R][R|E][T|O:R]")
graphviz.plot(d3)
```



d3 could be built by assigning different than in d2 direction between O and R (i.e. R->O) , because it is a node in p2 not having any direction (i.e. reversible)

5.4

Calculate the log-likelihood of the data given D2 and the log-likelihood of the data given D3. These values should be the same, explain why. You can use the `score` function with the argument `type = 'loglik`, or you can simply se the `logLik` function, which is just a wrapper for `score`. You dont need to provide paramter values for the CPDs of the DAG, `score` will estimate them for you. (1 point)

```
score(d2, survey, type="loglik")
```

```
## [1] -2350.686
```

```
score(d3, survey, type="loglik")
```

```
## [1] -2350.686
```

Calculated scores are same because conditional indpendencies extracted from data (“survey2.txt”) could be equally described by both graphs - d2 and d3.

Question 6: Modeling and Inference using Pyro (18 points)

If you are new to tensor-based frameworks, make sure you give yourself plenty of time for this question. It takes time to get used to debugging. One common source of bugs is integers, *pyro* prefers you use floats (e.g., `torch.tensor(1.0)` instead of `torch.tensor(1)`). If you hit a bug and solve it, why not share with your classmates on Piazza?

6.1 Modeling

```
import torch
import pyro
from pyro.distributions import Categorical
import torch
from collections import Counter

pyro.set_rng_seed(101)

prob_A = torch.tensor([0.36, 0.16, 0.48]) # A=adult, A=old, A=young

prob_S = torch.tensor([0.55, 0.45]) #S=F, S=M

prob_E = torch.tensor([
    [[0.64, 0.36], #E=high|S=F,A=adult, E=uni|S=F,A=adult
    [0.84, 0.16],  #E=high|S=F,A=old,   E=uni|S=F,A=old
    [0.16, 0.84]], #E=high|S=F,A=young, E=uni|S=F,A=young

    [[0.72, 0.28], #E=high|S=M,A=adult, E=uni|S=M,A=adult
    [0.89, 0.11],  #E=high|S=M,A=old,   E=uni|S=M,A=old
    [0.81, 0.19]], #E=high|S=M,A=young, E=uni|S=M,A=young
    ])

prob_O = torch.tensor([
    [0.98, 0.02], #O=emp|E=high,  O=self|E=high
    [0.97, 0.03]  #O=emp|E=uni,   O=self|E=uni
    ])


prob_R = torch.tensor([
    [0.72, 0.28], #R=big|E=high, R=small|E=high
    [0.94, 0.06]  #R=big|E=uni,  R=small|E=uni
    ])

prob_T = torch.tensor([
    [[0.71, 0.14, 0.15], #T=car|R=big,O=emp, T=other|R=big,O=emp, T=train|R=big,O=emp
    [0.68, 0.16, 0.16]], #T=car|R=big,O=self, T=other|R=big,O=self, T=train|R=big,O=self

    [[0.55, 0.08, 0.37], #T=car|R=small,O=emp, T=other|R=small,O=emp, T=train|R=small,O=emp
    [0.73, 0.25, 0.02]], #T=car|R=small,O=self, T=other|R=small,O=self, T=train|R=small,O=self
    ])

def transportation():
    S=pyro.sample("S", Categorical(probs=prob_S))
```

```
    A=pyro.sample("A", Categorical(probs=prob_A))
    E=pyro.sample("E", Categorical(probs=prob_E[S][A]))
    O=pyro.sample("O", Categorical(probs=prob_O[E]))
    R=pyro.sample("R", Categorical(probs=prob_R[E]))
    T=pyro.sample("T", Categorical(probs=prob_T[R][O]))
    return A, R, E, O, S, T
```

6.2.a Forward casual inference

```
samples_total = 10000
condtioned_on_E_uni = pyro.condition(transportation, data={"E": torch.tensor(1)})
samples_cond = [int(condtioned_on_E_uni()[5]) for i in range (samples_total)]

histogram_cond = Counter(samples_cond)
for key in histogram_cond:
    histogram_cond[key] /= samples_total

print(f'marginal distribution of T given E=uni, i.e. means of travel (0 - Car, 1 - Other, 2 - Train): {
```

```
## marginal distribution of T given E=uni, i.e. means of travel (0 - Car, 1 - Other, 2 - Train): Counte
```