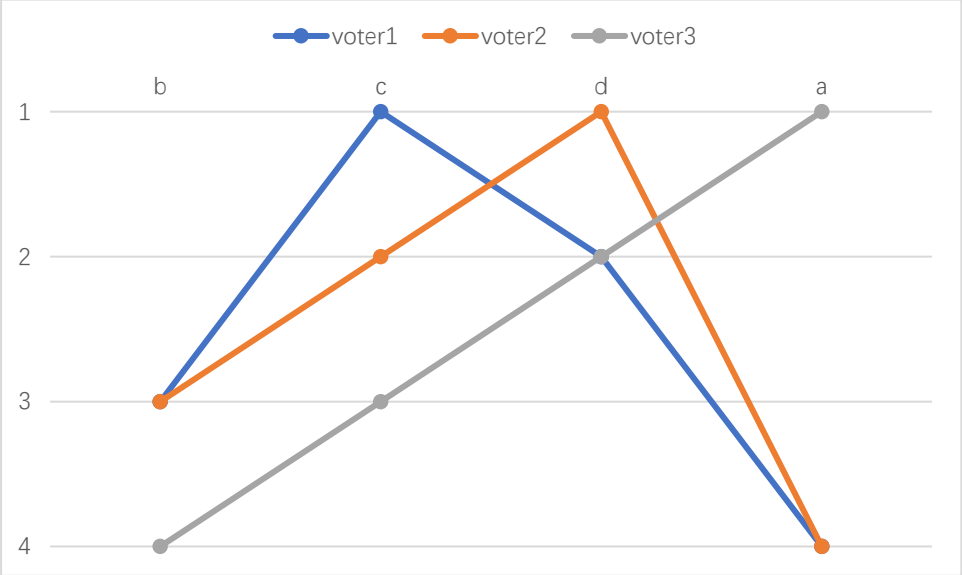
Name: Qiyao Zhou

Zid: z5379852

Question1:

1. True.



2.True

For voter1, c>d>b>a.

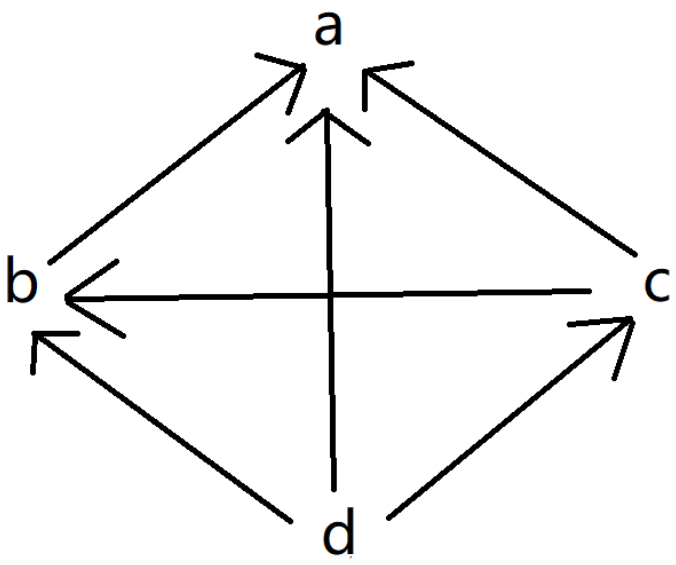
For voter2, d>c>b>a.

For voter3, a>d>c>b.

So: d>c:2-1=1 d>b:3 d>a:2-1=1(win)

So, d is the Condorcet winner.

3.



4.From the majority graph in 1.3, only d can reach every other alternative by a path d ->c->b->a. So, the top cycle set is {d}.

Question2

1. True.

Assume the allocation is not Pareto optimal.

Then there exists another allocation in which each agent gets at least as much utility and one agent strictly more utility.

But this means the allocation does not maximize utilitarian welfare which is a contradiction.

2. False.

Assume the allocation X which satisfies Pareto optimal is envy-free.

For all i, j ∈ N $X_i \succsim X_j$, $u_i\ (X_i) \geqslant u_i\ (X_j)$.

However, there is no allocation Y such that $u_i\ (Y_i) \geqslant u_i\ (X_i)$. for all i ∈ N.

So, if not all agents have monotonic and convex preferences, then an allocation which is both Pareto-efficient and envy-free will not exist.

3. True.

3.1 Assume that an allocation X is envy-free.

Then, for each i, j∈N, $X_i \succsim X_j$, $u_i\ (X_i) \geqslant u_i\ (X_j)$.

Thus, n*$u_i\ (X_i) \geqslant \sum u_i\ (X_j)$=$u_i$ (O).

Hence, $u_i\ (X_i) \geqslant u_i$ (O)/n.

3.2 Assume that an allocation X is proportionality.

Then, for each i∈N, $u_i\ (X_i) \geqslant u_i$ (O)/2.

Thus, $u_i\ (X_i)$ =u_i (O)/2.

So, for each i, j∈N, $X_i \succsim X_j$, $u_i\ (X_i) = u_i\ (X_j)$ which also satisfy envy-free.

4. False.

For example, there are 2 agents and 4 items:

The truth preference is:

1: a, b, c, d

2: b, c, d, a

For sequential allocation algorithm, 1 will get a and c while 2 get b and d.

But a can cheat with preference: b, a, c, d. In this case, 1 will get a and b while 2 get c and d.

Obviously, this is not strategyproof.

Question3

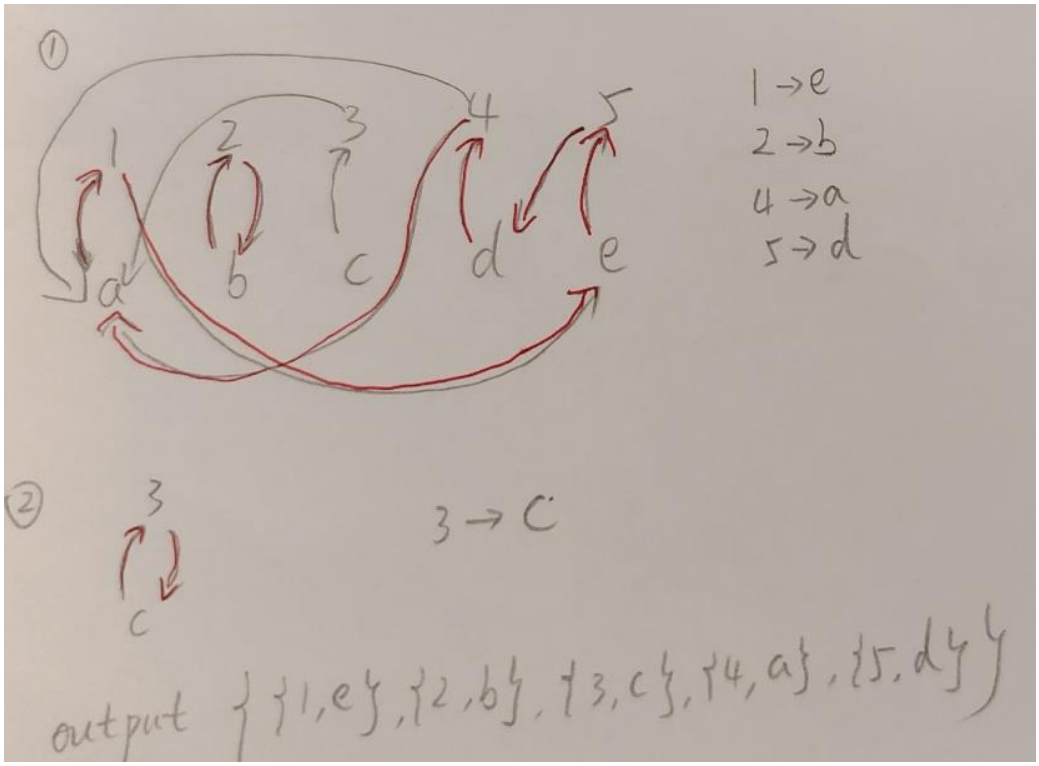
1.

- 3 and 4 apply to a; 2 applies to b; 5 applies to d; 1 applies to e;
- a rejects 3 in favour of 4;
- 3 applies to b;
- b rejects 2 in favour of 3;

- 2 applies to a;
- a rejects 4 in favour of 2;
- 4 applies to b;
- b rejects 4 in favour of 3;
- 4 applies to c and get accepted {{1,e},{2,a},{3,b},{4,c},{5,d}}

The resultant matching is Pareto optimal for the students.
{{1,e},{2,b},{3,a},{4,c},{5,d}} is a better allocation.

2.



3.

- Stability of the matching results obtained.
- Better allocation in terms of student welfare.
- Guiding students to tell the truth to get the best results.

4.

The output of TTC is efficient and TTC is strategy-proof.
For example:
Student: 1: b>a; 2: a; 3: a>b
School: a: 1>2>3; b: 3>1 each school has only one seat.
TTC algorithm:
Cycle: 1->b->3->a->1. So the output is {{1,b},{3,a}}.
It is obviously Pareto efficient and strategy-proof.

Question4

1.

```
rank(1,c,1).
rank(1,d,2).
rank(1,b,3).
rank(1,a,4).
rank(2,d,1).
rank(2,c,2).
rank(2,b,3).
rank(2,a,4).
rank(3,a,1).
rank(3,d,2).
rank(3,c,3).
rank(3,b,4).
```

2.

```
%Count number of alternatives and voters
alternativeNumber(N):-N=#max{K:rank(I,A,K)}.
voterNumber(Z):-Z=#max{I:rank(I,A,K)}.
alternative(X):-rank(I,X,K).

%Comparing the preferences of two alternatives at the same voter
compare(I,X,Y,1):-rank(I,X,K1),rank(I,Y,K2),K1<K2.
compare(I,X,Y,-1):-rank(I,X,K1),rank(I,Y,K2),K1>K2.

%Get the direction of the edges in the majority graph
major(1,X,Y,K):-compare(1,X,Y,K).
major(M+1,X,Y,K1+K2):-major(M,X,Y,K1),compare(M+1,X,Y,K2),alternativeNumber(N),M<N.
majority(X,Y):-major(M,X,Y,K),K>0,voterNumber(Z),M=Z.

%Output the condorcetWinner or UNSATISFIABLE
condorcetWinner(X):-alternative(X),A=#count{Y:majority(X,Y)},alternativeNumber(N),A=N-1.
:-A=#count{X:condorcetWinner(X)},A!=1.

#show condorcetWinner/1.
```

3.

```
%Count number of alternatives and voters
alternativeNumber(N):-N=#max{K:rank(I,A,K)}.
voterNumber(Z):-Z=#max{I:rank(I,A,K)}.
alternative(X):-rank(I,X,K).

%Comparing the preferences of two alternatives at the same voter
compare(I,X,Y,1):-rank(I,X,K1),rank(I,Y,K2),K1<K2.
compare(I,X,Y,-1):-rank(I,X,K1),rank(I,Y,K2),K1>K2.
```

```
%Get the direction of the edges in the majority graph
major(1,X,Y,K):-compare(1,X,Y,K).
major(M+1,X,Y,K1+K2):-major(M,X,Y,K1),compare(M+1,X,Y,K2),alternativeNumber(N),M<N.
majority(X,Y):-major(M,X,Y,K),K>0,voterNumber(Z),M=Z.

%Output the condorcetWinner or UNSATISFIABLE
condorcetWinner(X):-alternative(X),A=#count{Y:majority(X,Y)},alternativeNumber(N),A=N-1.

%Check if the correct condorcetwinner is generated
check(ncw):-A=#count{X:condorcetWinner(X)},A!=1.

%Compute the borda value of all alternatives
score(I,X,N-K+1):-check(ncw),rank(I,X,K),alternativeNumber(N).
total(1,X,K):-score(1,X,K).
total(M+1,X,K1+K2):-total(M,X,K1),score(M+1,X,K2),check(ncw),alternativeNumber(N),M<N.
borda(X,K):-total(M,X,K),voterNumber(Z),M=Z,check(ncw).

%Generate bordawinner comes first in alphabetical order
bordaMaxvalue(A):-A=#max{K:borda(_,K)},check(ncw).
bordaset(X):-borda(X,K),bordaMaxvalue(A),K=A,check(ncw).
bordaWinner(H):-H=#min{X:bordaset(X)},check(ncw).

#show condorcetWinner/1.
#show bordaWinner/1.
4.
%Count number of alternatives and voters
alternativeNumber(N):-N=#max{K:rank(I,A,K)}.
voterNumber(Z):-Z=#max{I:rank(I,A,K)}.
alternative(X):-rank(I,X,K).

%Comparing the preferences of two alternatives at the same voter
compare(I,X,Y,1):-rank(I,X,K1),rank(I,Y,K2),K1<K2.
compare(I,X,Y,-1):-rank(I,X,K1),rank(I,Y,K2),K1>K2.

%Get the direction of the edges in the majority graph
major(1,X,Y,K):-compare(1,X,Y,K).
major(M+1,X,Y,K1+K2):-major(M,X,Y,K1),compare(M+1,X,Y,K2),alternativeNumber(N),M<N.
majority(X,Y):-major(M,X,Y,K),K>0,voterNumber(Z),M=Z.

%Output the condorcetWinner or UNSATISFIABLE
condorcetWinner(X):-alternative(X),A=#count{Y:majority(X,Y)},alternativeNumber(N),A=N-1.

%Check if the correct condorcetwinner is generated
check(ncw):-A=#count{X:condorcetWinner(X)},A!=1.
```

```
%Compute the borda value of all alternatives
score(I,X,N-K+1):-check(ncw),rank(I,X,K),alternativeNumber(N).
total(1,X,K):-score(1,X,K).
total(M+1,X,K1+K2):-total(M,X,K1),score(M+1,X,K2),check(ncw),alternativeNumber(N),M<N.
borda(X,K):-total(M,X,K),voterNumber(Z),M=Z,check(ncw).

%Generate bordawinner comes first in alphabetical order
bordaMaxvalue(A):-A=#max{K:borda(_,K)},check(ncw).
bordaset(X):-borda(X,K),bordaMaxvalue(A),K=A,check(ncw).
bordaWinner(H):-H=#min{X:bordaset(X)},check(ncw).

%Get condorcetWinner or bordawinner as winner
winner(X):-condorcetWinner(X).
winner(X):-bordaWinner(X),check(ncw).

%Choose a voter
1 {voter(I):rank(I,A,K)} 1.

%Remove cases where the voter has the ultimate winner as their preference first
:-voter(I),rank(I,X,1),winner(X).

%Generate false ranking
chescore(X,A):-alternative(X),A=#count{Y:majority(X,Y)},alternativeNumber(N),A<=N-1,not
check(ncw).
chescore(X,K):-borda(X,K),check(ncw).
be(A):-voter(I),rank(I,A,X),rank(I,B,Y),X<Y,winner(B).
1 {better(X):be(X)} 1.
morethan(A,B):-chescore(A,K1),chescore(B,K2),K1>K2,better(C),A!=C,B!=C.
morethan(A,B):-chescore(A,K1),chescore(B,K2),K1=K2,A<B,better(C),A!=C,B!=C.
falserank(I,A,1):-voter(I),better(A).
falserank(I,A,N+1):-falserank(I,B,N),voter(I),alternativeNumber(M),N<=M-
1,alternative(A),C=#count{Y:morethan(A,Y)},C=N-1,better(D),A!=D.

%Generate true ranking by original preference
truerank(I,A,K):-rank(I,A,K),voter(I).

%Recomputing winner with false preferences using the previously used method
compare2(I,X,Y,1):-voter(I),falserank(I,X,K1),falserank(I,Y,K2),K1<K2.
compare2(I,X,Y,-1):-voter(I),falserank(I,X,K1),falserank(I,Y,K2),K1>K2.
compare2(I,X,Y,1):-rank(I,X,K1),rank(I,Y,K2),K1<K2,not voter(I).
compare2(I,X,Y,-1):-rank(I,X,K1),rank(I,Y,K2),K1>K2,not voter(I).
major2(1,X,Y,K):-compare2(1,X,Y,K).
major2(M+1,X,Y,K1+K2):-
```

```
major2(M,X,Y,K1),compare2(M+1,X,Y,K2),alternativeNumber(N),M<N.
majority2(X,Y):-major2(M,X,Y,K),K>0,voterNumber(Z),M=Z.
condorcetWinner2(X):-
alternative(X),A=#count{Y:majority2(X,Y)},alternativeNumber(N),A=N-1.
check2(ncw):-not condorcetWinner2(_).
score2(I,X,N-K+1):-check2(ncw),falserank(I,X,K),alternativeNumber(N),voter(I).
score2(I,X,N-K+1):-check2(ncw),rank(I,X,K),alternativeNumber(N),not voter(I).
total2(1,X,K):-score2(1,X,K).
total2(M+1,X,K1+K2):-
total2(M,X,K1),score2(M+1,X,K2),check2(ncw),alternativeNumber(N),M<N.
borda2(X,K):-total2(M,X,K),voterNumber(Z),M=Z,check2(ncw).
bordaMaxvalue2(A):-A=#max{K:borda2(_,K)},check2(ncw).
bordaset2(X):-borda2(X,K),bordaMaxvalue2(A),K=A,check2(ncw).
bordaWinner2(H):-H=#min{X:bordaset2(X)}.
falsewinner(X):-condorcetWinner2(X).
falsewinner(X):-bordaWinner2(X),check2(ncw).

%Excluding cases where the new winner is still not the voter's first choice
:-falsewinner(X),better(Y),X!=Y.

#show falserank/3.
#show truerank/3.
```