Part 1

Question 1:

a)

	start10	**start12**	**start20**	**start30**	**start40**
UCS	2565	Mem	Mem	Mem	Mem
IDS	2407	13812	5297410	Time	Time
A*	33	26	915	Mem	Mem
IDA*	29	21	952	17297	112571

b)

UCS: In terms of time, the Uniform Cost Search is the slowest, and in terms of memory usage, it runs out of memory except start10 (see table above). That's because UCS needs to spend time on expanding each node in increasing order of lower path cost. Both time and space complexity of UCS are all $O(b^{[C^*/\epsilon]})$, where C^* is the cost of the optimal solution, ϵ is the least cost of every transition. Thus, UCS is inefficient for both time and memory usage.

IDS: Iterative Deepening Search tries to combine the benefits of depth-first (low memory) and breadth-first (optimal and complete), so compared to UCS, this algorithm will not run out of the memory but will time out for the larger tests such as start30 and start40. The time complexity of IDS is $O(b^d)$ and its space cost is $O(bd)$. Thus, IDS is also inefficient in time but has better memory usage efficiency than UCS.

A*: A* Search combines the advantages of Uniform-Cost Search and Greedy Search, but it also keeps all expended node be stored, like UCS, so A* has memory usage inefficiency (see start30 and start40 in table above). The space complexity is $O(b^d)$. In terms of time, it is much quicker than UCS and IDS, the time complexity of A* depends on heuristic (heuristic is 0, the worst time complexity occurred, which is $O(b^d)$).

IDA*: Iterative Deepening A* is a low-memory variant of A*, it is the most efficient for both time and space cost among these four algorithms. Thus, the time cost is same as A*, which depends on heuristic too and $O(b^d)$ is also the time complexity of the worst case. However, the space complexity is less than A*, it is $O(bd)$. Thus, IDA* has efficiency on both time and space.

Question 2:

(a) & (c)

	start50		**start60**		**start64**	
IDA*	50	14542612	60	321252368	64	1209086782
1.2	52	191438	62	230861	66	431033
1.4	66	116342	82	4432	94	190278
1.6	100	33504	148	55626	162	235848
Greedy	164	5447	166	1617	184	2174

(b)

According to the function provided from week 2 tutorial (Question 5), $f(n) = (2 - w) \times g(n) + w \times h(n)$, where $0 \leq w \leq 2$. So, in the code, I set W is 1.2 firstly, and then replace the original code "F1 is G1 + H1" (see left picture below) to "F1 is (2 – W) * G1 + W * H1" (see right picture below).

```
41      G1 is G + C,
42      h(Node1, H1),
43      F1 is G1 + H1,
44      F1 =< F_limit,
45      depthlim([Node|Path],
46
```

```
42      h(Node1, H1),
43      W is 1.2,
44      F1 is (2 - W) * G1 + W * H1,
45      F1 =< F_limit,
46      depthlim([Node|Path], Node1, G
```

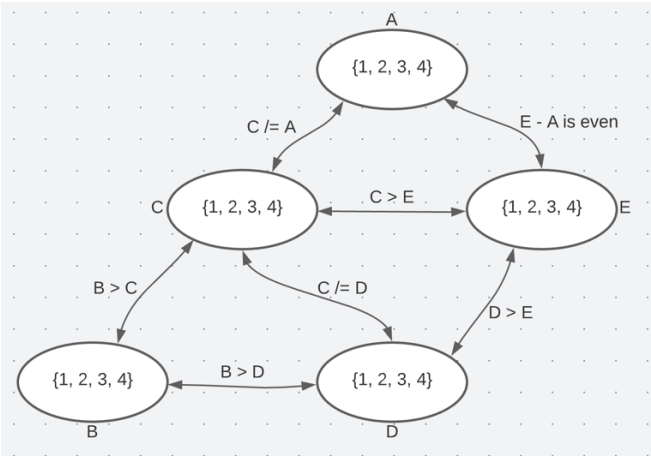
(d)

The formula "$f(n) = (2 - w) \times g(n) + w \times h(n)$, where $0 \leq w \leq 2$" applies to all five algorithms. When w = 0, it is UCS/IDS; when w = 1, it is A*/IDA*; and when w = 2, it is greedy. To compared time complexity of five algorithms, greedy search is the fastest, next is A*/IDA*, last is UCS/IDS. And form the table above, we can see the value of w is gradually increase (from 1 to 2), the G (length of path) increases whereas the N (the total number of states expanded) decreases, so the slower an algorithm is, the better qualities of solutions have. The example from table above, IDA*, with smaller w value, produce shorter path; greedy search, with bigger w value, produce longer path than IDA* does. Thus, when w is increasing from 1 to 2, the algorithm becomes faster, but the qualities of solutions become worse.

Part 2

Question 1:

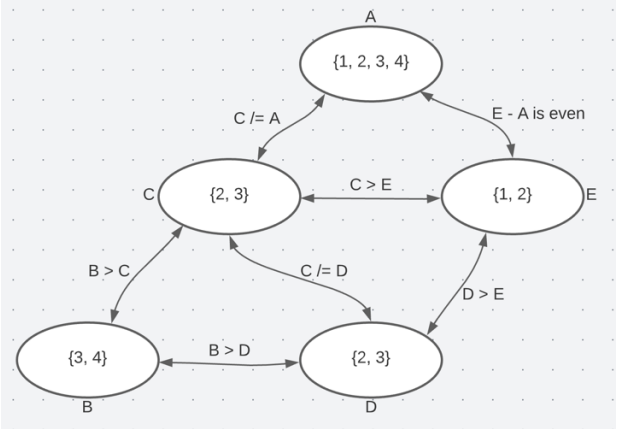
- Draw the constraint graph



- Show which elements of a domain are deleted at each step, and which arc is responsible for removing the element.

Arc	Relation	Value(s) Removed
<C, B>	B > C	**C = 4**
<B, C>	B > C	**B = 1**
<D, B>	B > D	**D = 4**
<D, E>	D > E	**D = 1**
<E, D>	D > E	**E = 4 && E = 3**
<C, E>	C > E	**C = 1**
<B, C>	B > C	**B = 2**
Stopped		

- Show explicitly the constraint graph after arc consistency has stopped.



- Show how splitting domains can be used to solve this problem. Include all arc consistency steps.

  Cause C and D has same domain {2, 3} until before (see picture above), so we need to consider two cases.

  Case 1:

  - Splitting the domain of D firstly
  - When D is 2,

Arc	Relation	Value(s) Removed
<C, D>	C /= D	**C = 2**
<B, C>	B > C	**B = 3**
<E, D>	D > E	**E = 2**
<A, E>	E – A is even	**A = 2 && A = 4**
<A, C>	C /= A	**A = 3**
Stopped		

  Thus, the solution of CSP problem is A = 1, B = 4, C = 3, D = 2, E = 1.

  Case 2:

  - Splitting the domain of C firstly
  - When C is 2,

Arc	Relation	Value(s) Removed
<D, C>	C /= D	**D = 2**
<B, D>	B > D	**B = 3**
<E, C>	C > E	**E = 2**
<A, E>	E – A is even	**A = 2 && A = 4**
Stopped		

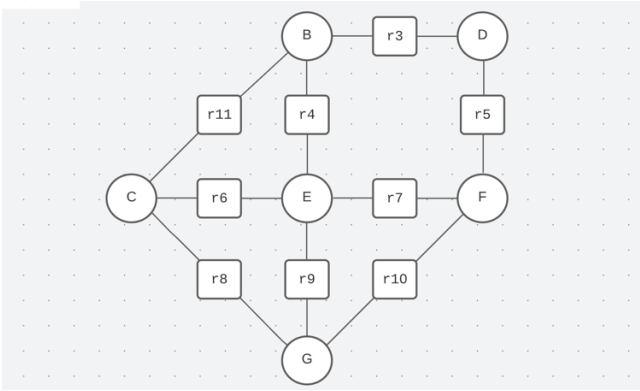
- Then splitting the domain {1, 3} of A
- When A is 1, the solution of CSP problem has got, which is A = 1, B = 4, C = 2, D = 3, E = 1.

In conclusion, there are two solutions of this CSP problem, the first one is “A = 1, B = 4, C = 3, D = 2, E = 1”, another one is “A = 1, B = 4, C = 2, D = 3, E = 1”.

Question 2:

(a)

According to the lecture, for eliminating variable A, the constraint r_1 and the constraint r_2 are removed, and constraint r_{11} created between variable B and variable C (see picture below).



(b) For subsequently eliminating variable B, the constraint r_3, the constraint r_4 and the constraint r_{11} are all removed. The constraint r_{12} created between variable C and variable D, and constraint r_{13} created between variable D and variable E (see picture below).