Programming Assignment 3

CSP : standard, forward checking, arc consistency

2012003567

이상화

**1. 문제**

N-Queen 문제를 standard csp, forward checking, arc consistency를 이용해서 해결한다.

**2. 문제 해결 과정**

CSP로 문제를 해결할때는 기본적으로 DFS를 사용하므로 과제 1번에서 사용한 DFS클래스를 그대로 가져와서 사용하였다. 일반적인 State space search와 다른 점은 CSP의 경우 value를 assign한 뒤 이 value가 지금까지 assign된 value들과 consistency를 유지하는지를 체크하고 consistent하지 않다면 assign은 undo하고 backtracking을 한다. 즉, 답이 나올 수 없는 path인 경우에는 complete하게 assign하지 않고 돌아가서 다른 path에서 search를 진행하게 된다.

forward checking까지 하게 되는 경우 assign되지 않은 variable에 대해서도 답이 나올수 있는지 없는지를 체크한다. 예를 들어, 4-Queen 문제에서 0번 column 0번 행에 queen을 assign 했다면 1,2,3번 colum이 가질 수 있는 domain의 개수는 4개에서 2개로 줄어들게 된다. 이런식으로 변수를 assign한 뒤 그 변수가 공격하는 assign되지 않은 column의 domain을 unable로 만들고 만약 able한 domain의 개수가 0개라면 assign을 undo하고 backtracking을 한다.

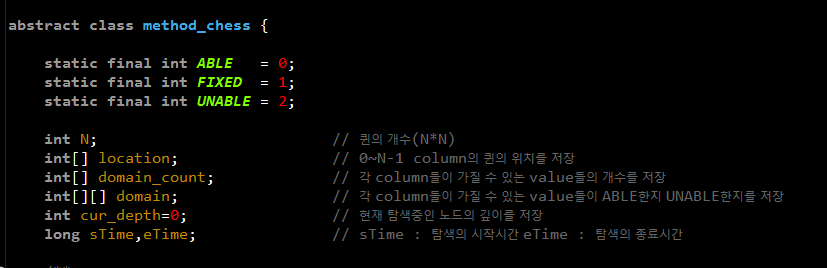
arc consistency checking까지 할 경우 forward checking만 하는 것이 아니라 forward checking으로 인해 domain이 변화된 column에 의해 domain이 변화된 column들까지 domain을 update해준다. 예를 들어 i번째 column에 assign을 하게 되면 i+1~N번째 column은 전부 domain이 변하게 되는데 이 변한 domain들에 영향을 받는 column들의 domain을 계속해서 update해주어서 domain의 변화가 없을 때까지 진행한다. arc consistency checking을 마친 뒤에 able한 domain의 개수가 0인 column이 있으면 assign을 undo하고 backtracking을 한다.

**3. 코드 분석**

**(1) method\_chess class**

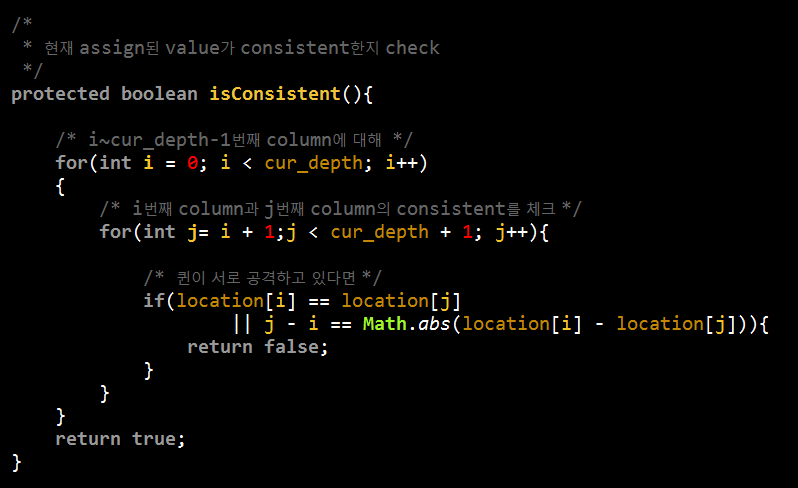
method\_chess class는 CSP\_std\_chess,CSP\_fwd\_chess,CSP\_arc\_chess에서 공통으로 사용하는 변수와 메소드들을 모아놓은 class이다.

➀ 공통변수



N \* N 배열인 domain은 각 column이 가질 수 있는 domain value들이 ABLE인지 FIXED인지 UNABLE인지를 저장해놓는 배열이다. domain\_count배열은 domain의 각 column의 중 ABLE인 value의 개수를 저장해놓는 배열이다. 이렇게 따로 count해주는 배열을 따로 만든 이유는 forward checking이나 arc consistency를 checking할 때 domain 배열을 순회하면서 ABLE의 개수를 count하게 되면 시간이 너무 오래 걸리게 되므로 ABLE->UNABLE 될때마다 count를 1씩 감소시켜줌으로써 개수를 따로 유지하도록 하였다.

➁ isConsistent()



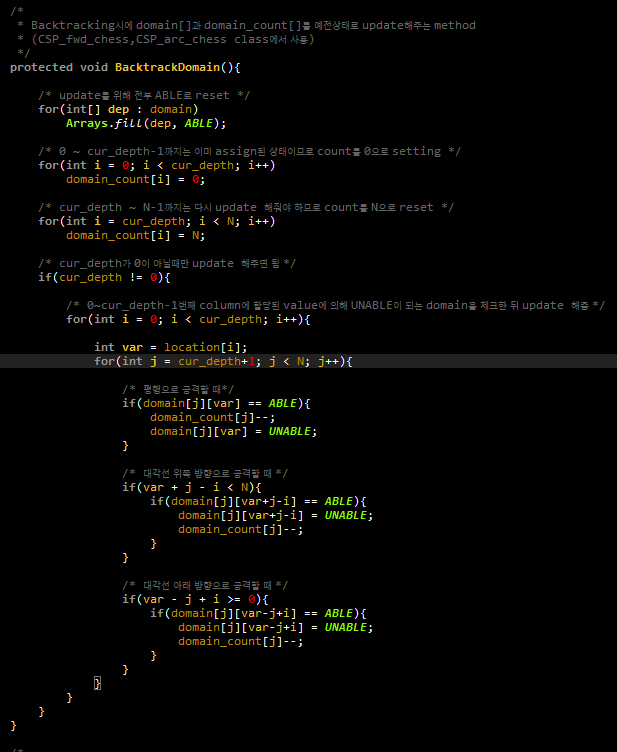
isConsistent 메소드는 cur\_depth번째 column에 queen을 assign했을 때 consistent한지 안한지를 체크하는 메소드이다. 이 메소드는 Standard CSP뿐만 아니라 forward checking,arc consistency에서도 사용되어지는 공통 메소드이다. 현재 0~cur\_depth번째 column까지 queen이 assign된 상태에서 이 퀸들이 서로 공격하는지 안하는지를 체크해서 공격을 하면 false를 return하고 공격하지 않으면 true를 return하게 된다.

➂ getNewDomain()



getNewDomain 메소드는 Forward checking과 arc consistency checking에서 사용된다. 이 함수는 어떤 cur\_depth column에 assign이 되었을 때 그것으로 변화되는 domain,domain\_count 테이블을 update해주는 메소드이다. 인자로 클래스 자체의 domain,domain\_count를 전달하는 것이 아니라 임시로 만든 new\_domain,new\_domain\_count를 전달한다. 그 이유는 만약 새로운 domain에서 consistent하지 않다는 것이 발견되어지면 이전 domain 테이블로 돌아가야 하기 때문에 클래스의 domain,domain\_count를 덮어 씌우는 것이 아니라 새로운 테이블을 만들어서 복사를 해준 뒤에 update된 domain을 구한다. cur\_depth번째 column에 대해서는 현재 할당한 row var(domain[cur\_depth][var])에는 FIXED로 체크해주고 나머지 row에는 UNABLE을 체크해준다. 나머지 unassign된 column (cur\_depth+1)~(N-1)의 domain은 현재 domain[cur\_depth][var]에게 공격받는 queen의 domain을 UNABLE로 만들어주고 domain\_count를 1씩 감소시켜 준다. 이 때 이미 UNABLE로 되어있는 위치에 대해서는 column의 domain\_count를 감소시켜서는 안된다.

➃BacktrackDomain()

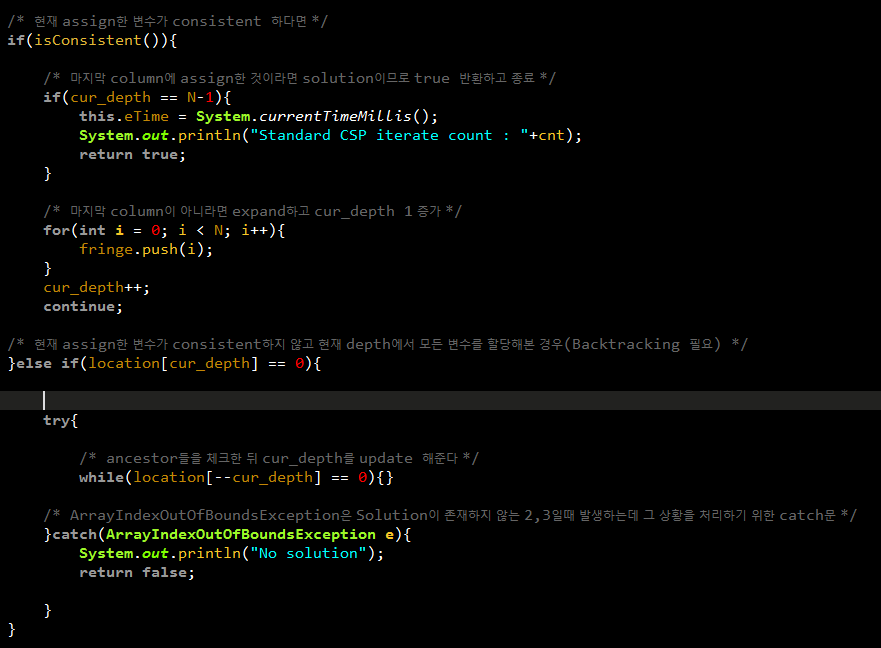


BacktrackDoamin method는 backtracking을 할 때 각 column의 domain을 예전 상태로 만들어주는 method이다. 재귀로 구현했으면 backtracking시에 예전 domain table 을 유지하는 것이 쉬웠을 것 같은데 Stack을 이용해서 구현했기 때문에 domain과 domain\_count table을 초기화 시킨 뒤에 0~cur\_depth-1번째 column에 assign된 queen에 의해 공격당하는 cur\_depth~N-1번째 column의 row들을 UNABLE로 만들어주는 방식으로 domain table을 복구해주었다.

**(1) CSP\_std\_chess class**

Standard CSP를 이용해서 문제를 푸는 class이다.

➀ solve() method logic



CSP\_std\_chess class의 solve method의 while loop의 주요부분인 if else문이다. 먼저 위의 상황은 cur\_depth번째 column에 queen을 어떤 위치에 assign한 상황이다. 이 상황에서 if(isConsistent())를 통해 assign뒤에도 consistent한지를 체크한다.

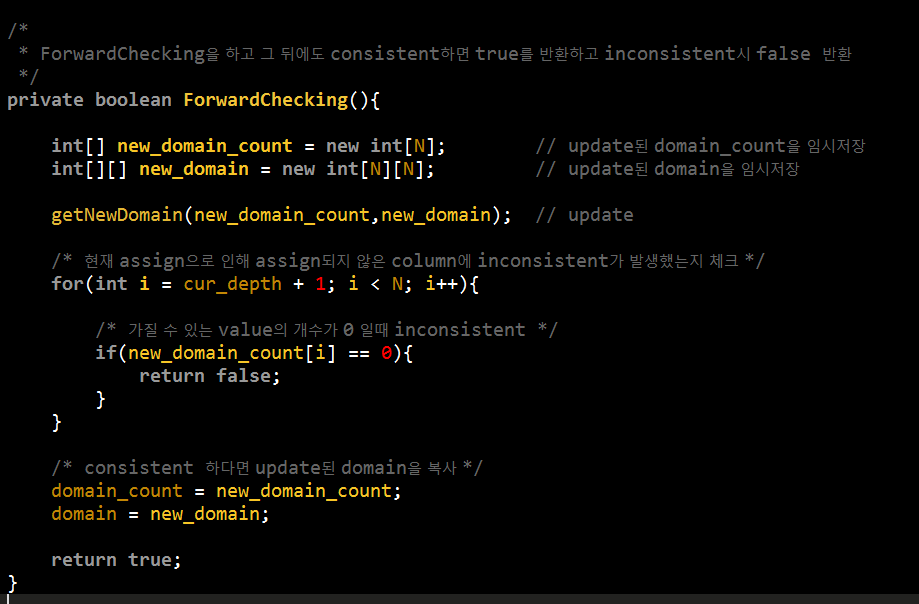
consistent한 경우에 cur\_depth가 N-1인 경우 즉 모든 column에 queen을 assign한 상황이라면 solution이므로 true를 반환하고 종료한다. N-1이 아니라면 cur\_depth를 1 증가시키고 expand시켜준뒤 continue한다.

inconsistent한 경우에는 backtracking을 해야하는데 else if(location[cur\_depth] == 0) 조건을 걸어준 이유는 location[cur\_depth] != 0인 경우에는 아직 cur\_depth에서 assign해볼 queen의 value가 남아 있는 상태이므로 그대로 while문의 처음으로 돌아가 fringe 첫번째 값을 cur\_depth에 할당해주면 된다. location[cur\_depth] == 0인 경우는 cur\_depth에서 모든 value를 assign해본 상태이므로 backtracking을 해야 한다. 그러므로 어느 depth까지 backtrack을 해야 하는지 체크해준 뒤 while문의 처음으로 돌아간다. try~catch문을 추가해준 이유는 Solution이 존재하지 않는 N=2,3일때 발생하는 exception을 처리해 주기 위해서이다.

**(2) CSP\_fwd\_chess**

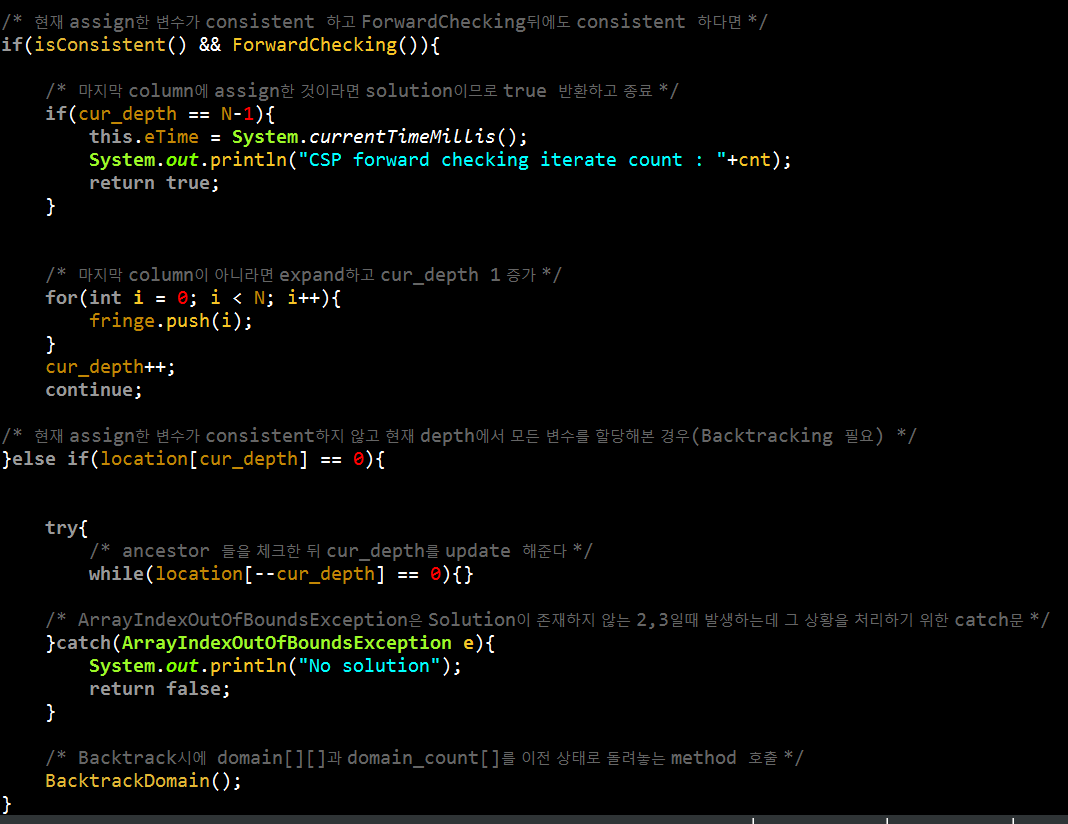
Forward checking with CSP으로 문제를 해결하는 class이다.

➀ ForwardChecking()



ForwardChecking을 하고 consistent하면 true, inconsistent하면 false를 반환하는 메소드이다. 먼저 cur\_depth에 assign된 queen에 의해 update된 domain과 domain\_count를 구해야 한다. false return시에 domain을 그대로 유지해야 하므로 domain과 domain\_count를 덮어씌우지 않도록 임시 변수 new\_domain과 new\_domain\_count를 선언해주고 getNewDomain메소드에 두 변수를 전달해 update돈 domain을 구한다. 그 뒤에 만약 (cur\_depth+1)~(N-1)번째 column중 new\_domain\_count가 0인 column이 존재하면 inconsistent 하다는 의미이므로 false를 return해준다. 만약 consistent하다면 domain과 domain\_count를 update된 domain으로 할당해주고 true를 return 한다.

➁ solve() method logic

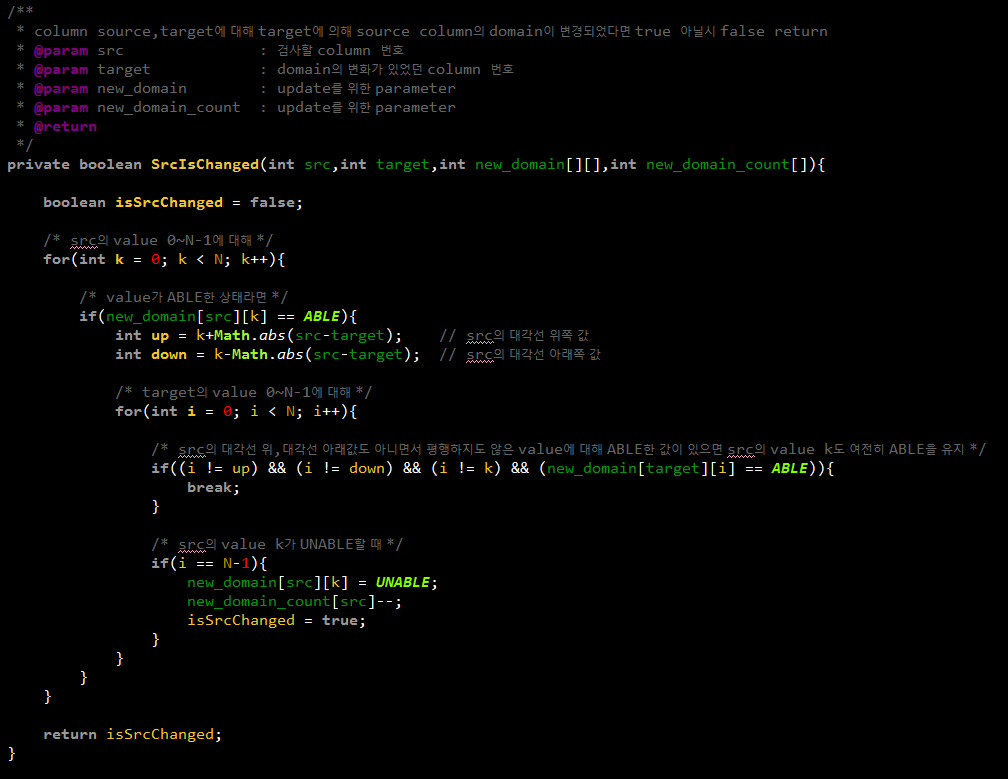


CSP\_std\_chess의 solve()와 다른 점은 두가지 밖에 없다. 첫번째 다른점은 if문에서 isConsistent()에 ForwardChecking()조건도 추가해 둘다 true여야지만 탐색을 계속 진행하게 한다. 두번째 다른점은 Standard CSP와는 다르게 domain과 domain\_count 테이블을 유지하고 있으므로 backtracking을 의미하는 else if문에서 마지막에 BacktrackDomain()을 호출해줌으로써 domain과 domain\_count 테이블을 예전상태로 되돌려준다는 점이다.

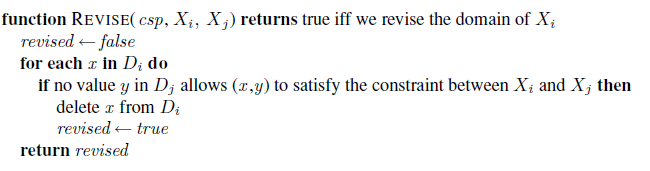
**(3) CSP\_arc\_chess**

Arc Consistency를 이용해 문제를 해결하는 class이다.

➀ SrcIsChanged()



SrcIsChanged 메소드는 교재에 나오는 AC-3 알고리즘에 이용되는 아래의 REVISE 함수를 NQueen문제에 맞게 변형한 함수이다.

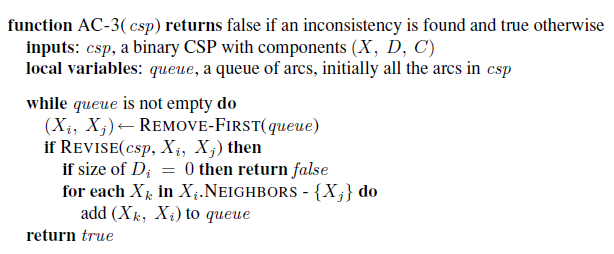


REVISE 함수에서의 Xi는 SrcIsChanged에서의 src가 되고 Xj는 target이 된다. 즉 target에 column에 의해 src column의 domain이 변경되면 변경된 사항을 new\_domain과 new\_domain\_count에 적용해주고 변경되었으면 true, 아니면 false를 return 해준다.

➁ ArcConsistencyChecking()



ArcConsistencyChecking 메소드는 교재에 나오는 아래의 AC-3 알고리즘을 NQueen 문제에 맞게 변형한 것이다.



Queue를 구현하기 위해 LinkedList를 이용하였다 이 Queue에 처음에는 현재 cur\_depth column에 assign된 queen에 의해 domain의 변화가 생길 수 있는 column의 번호를 넣어주었는데 이 column은 (cur\_dep+1)~(N-1)이 된다. 왜냐하면 assign되지 않은 모든 column은 assign된 queen에 의해 적어도 같은 열에 있는 value는 UNABLE이 되기 때문이다. 그 뒤 while문에서 Queue에서 값을 하나씩 꺼내서 Queue에서 나온 column에 의해 domain이 영향을 받을 수 있는 column들을 전부 SrcIsChanged 메소드를 이용해서 체크를 해주는데 이것들도 역시 (cur\_depth+1)~(N-1)의 column들일 수 밖에 없다. domain\_count가 0이 되는 column이 발견되면 inconsistent한 assign이었다는 의미이므로 false를 return하며, Queue가 빌때까지 false를 return하지 않았다면 consistent하다는 의미이므로 domain과 domain\_count를 덮어씌우고 true를 return 한다.

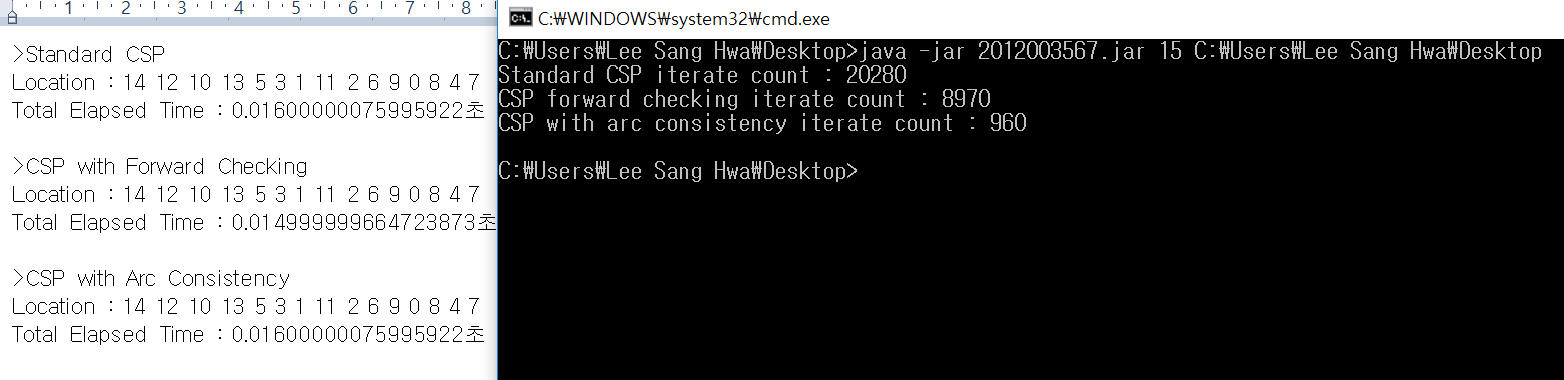
➂ solve() method logic



CSP\_fwd\_chess class의 solve()에서 if문의 ForwardChecking() 대신 ArcConsistencyChecking()이 들어갔다는 것을 제외하고는 완전히 일치하는 코드이다.

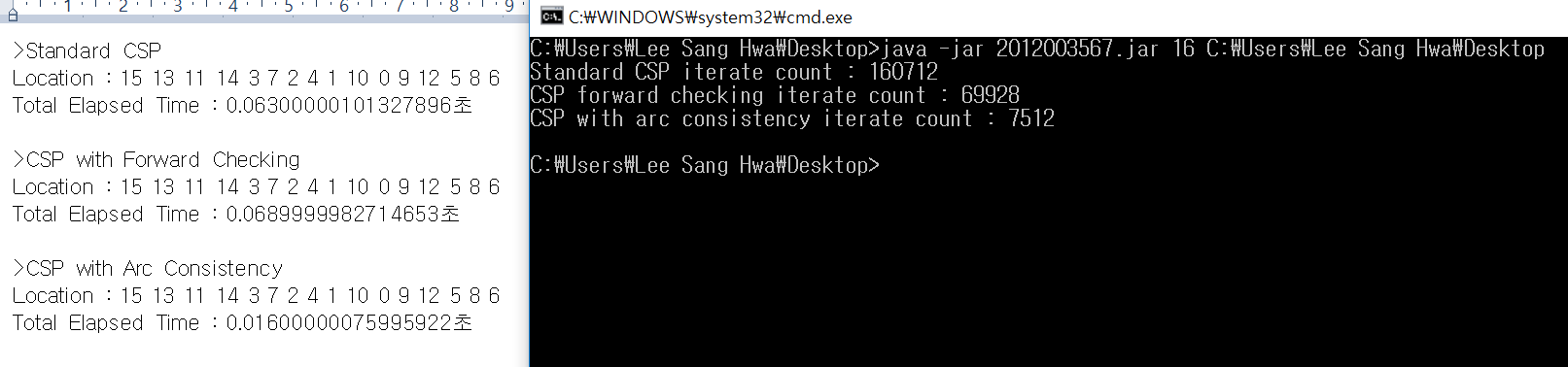
**4. 실행 결과**

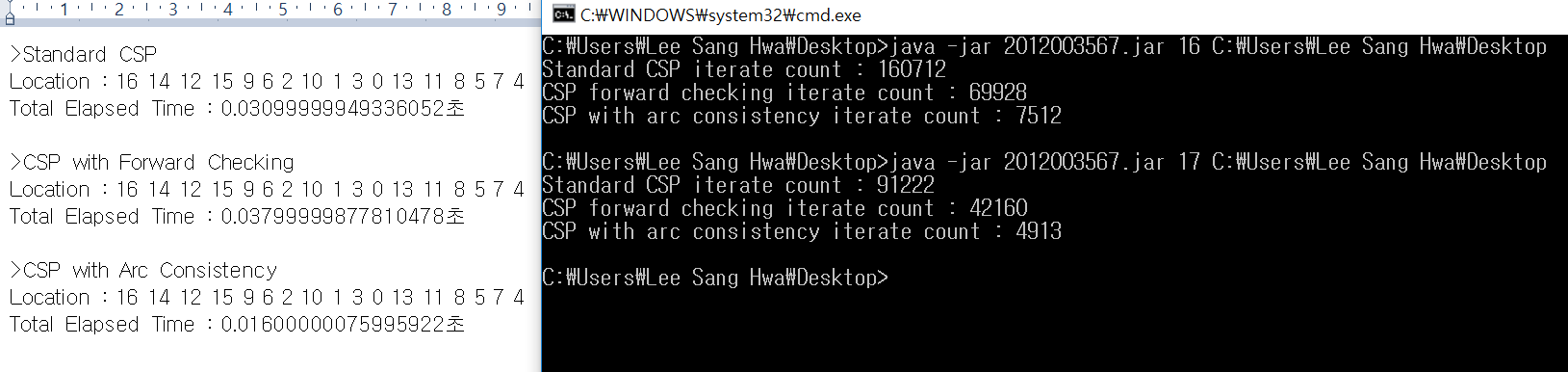
**(1) 1 <= N <= 15**

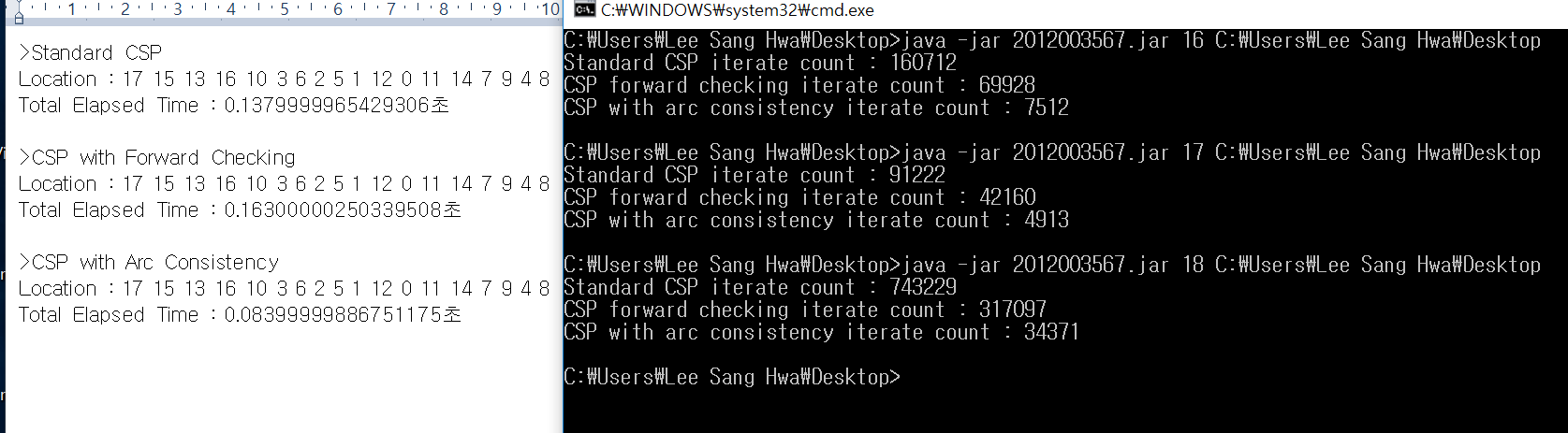


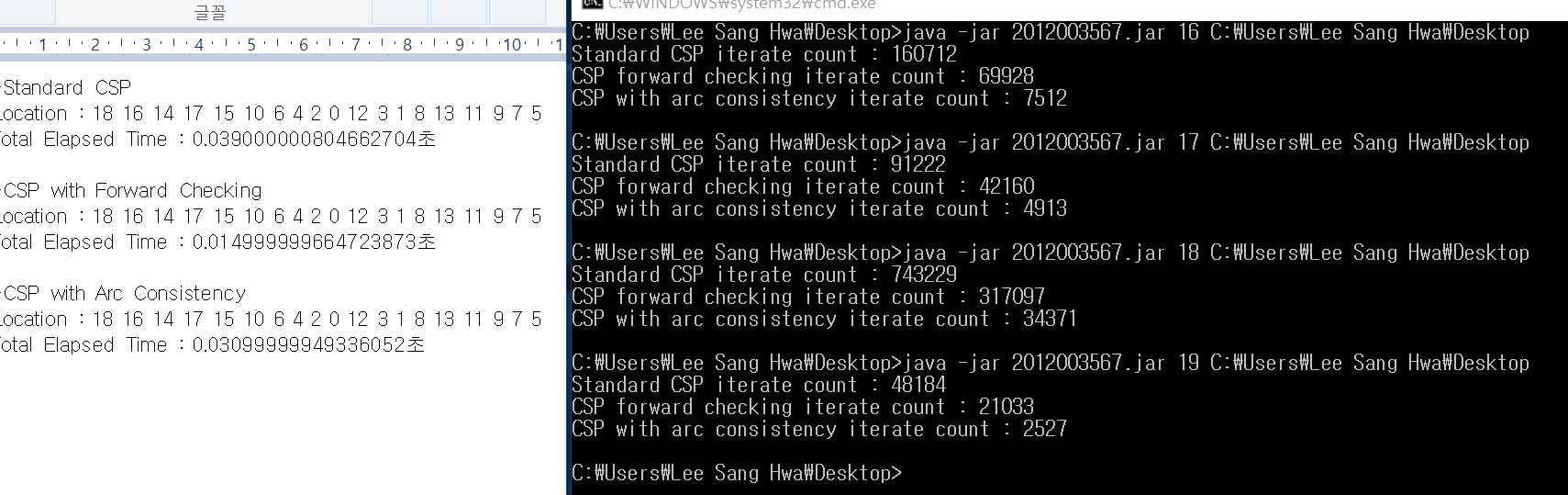
정확한 분석을 위해 각 메소드의 iteration 횟수도 출력해주었다. N=15일 때까지의 속도는 여러 번 측정해봐도 위와 같이 비슷하게 나왔다. CSP,Forward,Arc 순으로 iteration 횟수가 훨씬 줄어들긴 하지만 워낙 빠른 시간에 연산이 끝나서 정확한 측정이 안되는 것 같다.

**(2) 15 < N < 20**



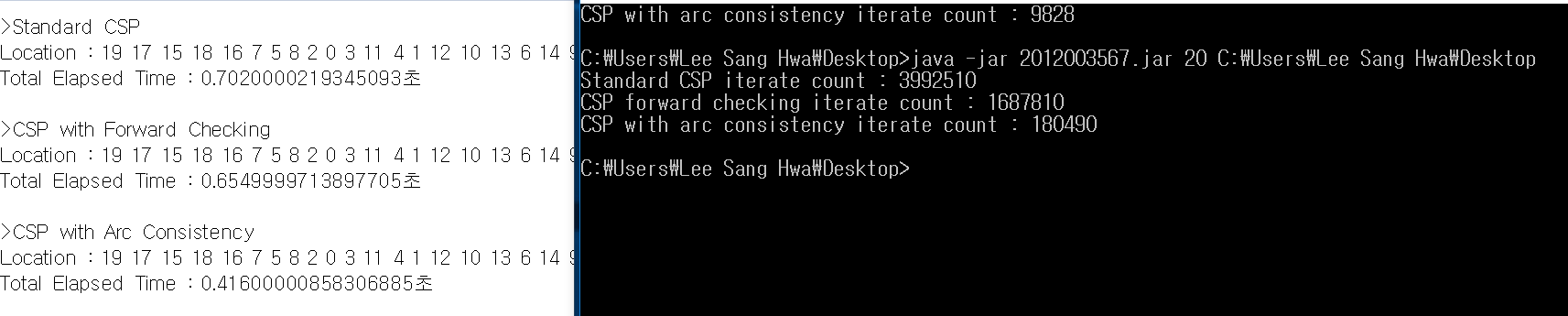


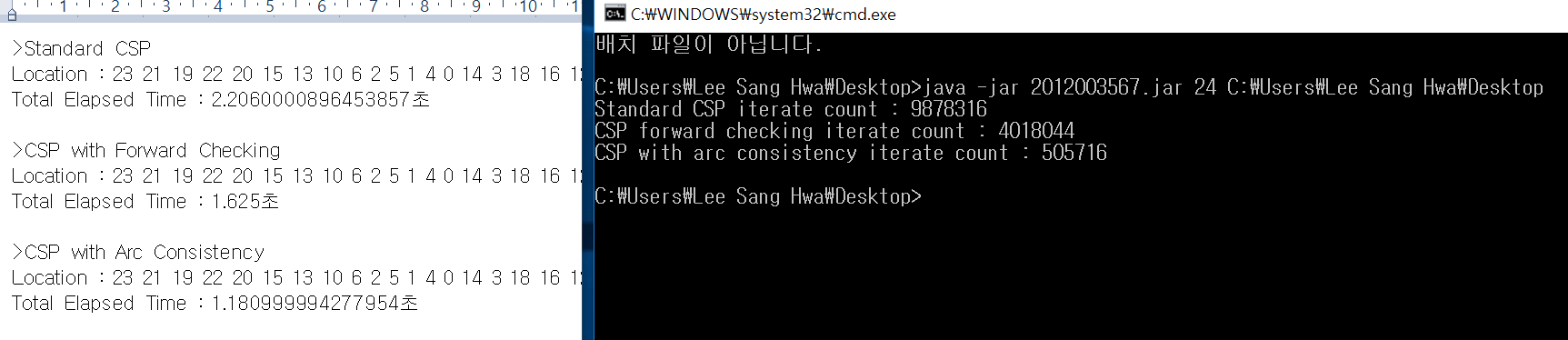


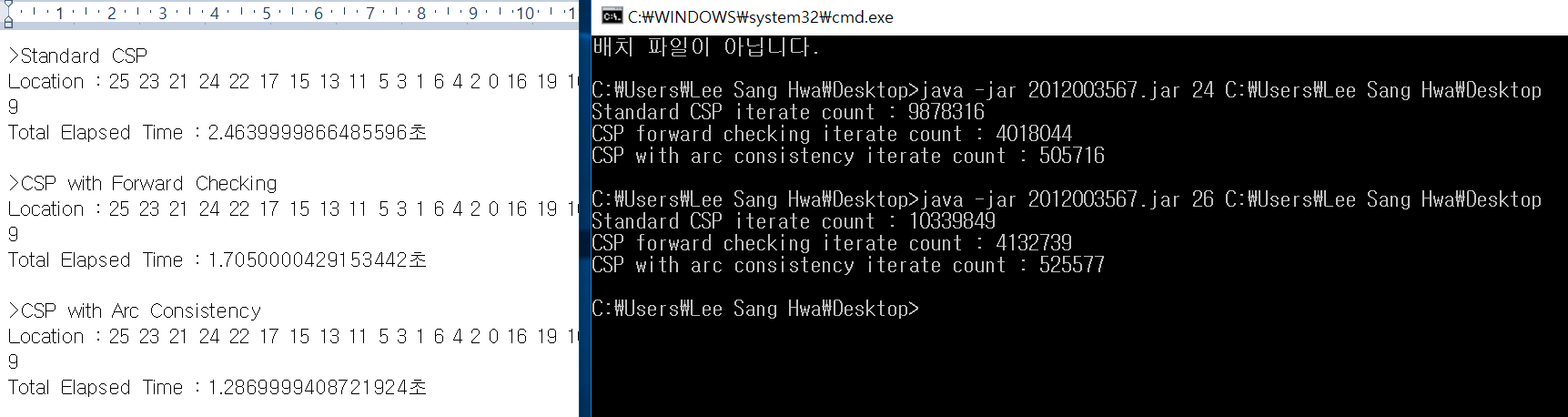


16~19일때의 실행 결과이다. 속도가 Arc Consistency > Standard CSP > Forward Checking의 순으로 나오고 있다. 개념적으로는 Arc Consistency > Forward Checking > Standard CSP이렇게 나와야 할 것 같지만 기대한 결과가 나오지 않았다. 원인은 Forward Checking 연산시 발생하는 overhead가 Forward Checking으로 인해 줄어드는 iteration에 의한 효과보다 더 크기 때문이다. 구현시 domain과 domain\_count를 복사하는 과정에서 overhead가 너무 심하게 발생한 것 같다. 반면 Arc Consistency의 경우 배열을 복사하는 overhead뿐만 아니라 계속해서 column들의 위치를 비교하고 domain을 update하는 overhead까지 발생하지만 iteration count를 어떤 경우에는 standard csp에 비해 1/200배 만큼이나 줄여주기 때문에 overhead를 상쇄할 수 있었던 것 같다.

**(3) N >= 20**

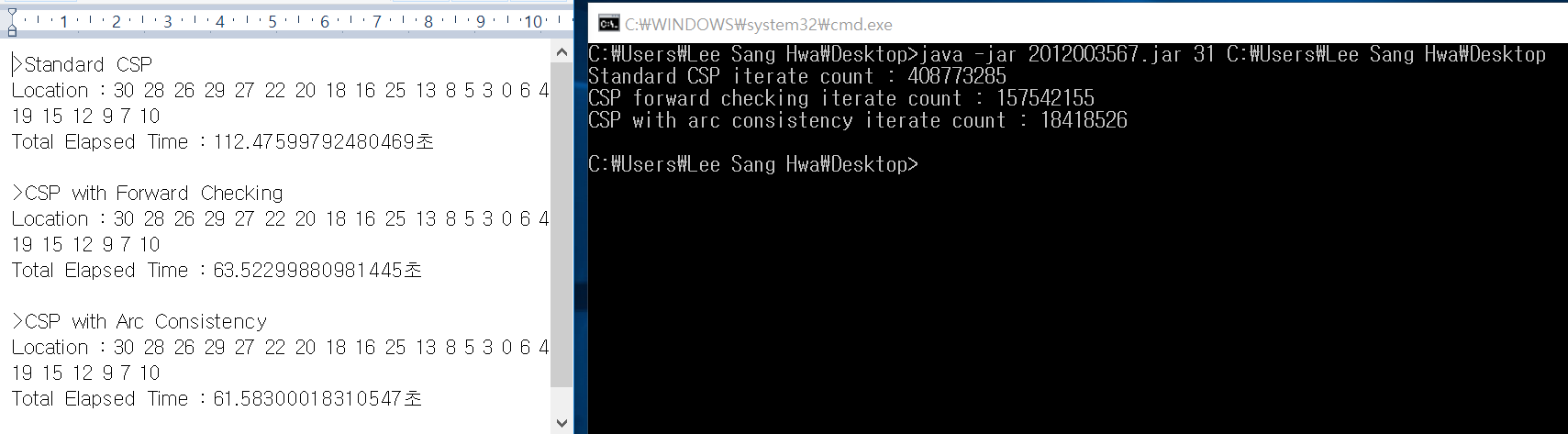






N>=20 인 경우부터는 속도가 Arc Consistency > Forward Checking > Standard CSP 순으로 나온다. Forward Checking으로 인한 iteration 횟수 감소의 효과가 overhead를 뛰어넘은 것 같다. N이 커질수록 iteration 횟수의 영향을 더 많이 받는 것 같다.

**(4) 1~2 분 이내에 측정 가능한 최대 N = 31**



N = 30일때는 모든 메소드가 2분 이상 시간이 걸렸지만 N = 31 일때는 2분 이내에 답을 출력할 수 있었다. N = 30일때보다 왜 N = 31일때가 더 빠른지는 그림까지 그려보면서 생각해보려고 했는데 분석하는데에 실패하였다. Standard CSP보다 Forward Checking과 Arc Consistency가 두배정도 빠르게 답을 출력했다. N이 커질수록 속도차가 커지는 것 같은데 Forward Checking과 Arc Consistency는 크게 차이가 안나는 것 같다. AC-3 알고리즘의 시간 복잡도는 n^2\*d^3의 만만치 않은 시간복잡도와 비효율적인 구현이 원인인 것 같다. 가장 최적화된 방법으로 AC-3를 구현하면 어느정도 속도가 나올지가 궁금하다.