

2008012647 장현준

pkthads@gmail.com

Artificial Intelligent HW3

Table of Contents

[Problems 2](#_Toc386571797)

[Develop Environment 2](#_Toc386571798)

[Source File Information 2](#_Toc386571799)

[Main Process 2](#_Toc386571800)

[Step 1. Generating MAP INFORMATION 2](#_Toc386571801)

[Step 2. Start Map Coloring 4](#_Toc386571802)

[Step 2-1. Back Tracking only 4](#_Toc386571803)

[Step 2-2. Back Tracking with Forward Check 4](#_Toc386571804)

[Step 2-3. Back Tracking with MAC (Maintaining Arc Consistency) 4](#_Toc386571805)

[Step 3. Show result on Window Screen. (Option) 5](#_Toc386571806)

[Result and Comment 5](#_Toc386571807)

[1. Option & Result 5](#_Toc386571808)

[2. Comment 8](#_Toc386571809)

[Future works (Remaining Problems) 8](#_Toc386571810)

[1. Give it up too fast 8](#_Toc386571811)

[2. Use Too much Time 9](#_Toc386571812)

# Problems

Generate random instances of map-coloring problems as follows: scatter points on the unit square; select a point at random, connect by a straight line to the nearest point such that is not already connected to and the line crosses no other line; repeat the previous step until no more connections are possible. The points represent regions on the map and the lines connect neighbors. Now try to find -colorings of each map, for both and using backtracking, backtracking with forward checking, and backtracking with MAC (maintaining arc consistency). Construct a table of average run times for each algorithm for values of up to the largest you can manage. Comment on your result.

# Develop Environment

Program: Microsoft Visual Studio Express 2012 for Windows Desktop

Library: OpenGL (freeglut). Need to Link freeglut.lib

# Source File Information

**main.cpp**: 메인 소스 파일로, 여러 가지 Boolean 옵션들을 정의한다. (boolLog, boolSave…)

**globals.h**: 전반적으로 사용하는 전역 변수들과 Point와 Line의 구조체를 선언한다.

**MapCreator.cpp**: 맵을 생성하는 소스로 Points와 Lines 의 구조체에 정보를 저장한다.

**MapColoring.cpp**: 주어진 3가지 방법을 이용하여 Points에 Color값들을 저장한다.

**MapScene.cpp**: OpenGL을 이용하여 결과를 윈도우 화면으로 보여준다.

# Main Process

## Step 1. Generating MAP INFORMATION

Map을 생성하는 기본적인 과정은 아래와 같다.

1. 의 좌표 계 위에 랜덤으로 중복되지 않는 점을 개 찍는다.
2. 생성된 점 중에서 랜덤으로 한 점 을 뽑고 이 점과 가장 가까운 점 를 선택한 다음, 이 점이 다른 점들과 만나는지 확인한다(). 만약 교차하는 선이 없다면 선에 대한 정보를 양 점 과 에 입력한다. 이 과정을 충분히 많이 만큼 반복한다.
3. 랜덤으로 생성된 정보를 기반으로, 하나의 점씩 순차적으로 생성 가능한 Line을 만들고 그 정보를 양 점에 저장한다. 이를 통해 모든 생성 가능한 Line이 생성된다.

위 과정들은 각각 MapCreator.cpp 의 , , 함수에 구현되어있다. 만 실행하였을 경우, Nearest한 Point에 대해서만 Line을 생성하려고 하기 때문에 모든 선이 그려지지 않는 문제가 있었다. 그래서 3번 과정을 추가하였다. 하지만 를 실행하여도 모든Line 이 생성되지 않는 문제가 있지만, 충분히 많은 Line 이 생성되므로 문제는 없다고 생각한다.

Line이 교차하는지 확인하는 함수는 이다. 이는 각 점들의 직교하는 성질을 이용하여 판단하는데, 먼저 인 경우 선분 와 는 같은 기울기를 갖는다. 이 경우 두 선분이 하나의 직선 위에 있는지 확인할 필요가 있다.

또, 각 점 와 에 대해 선분 선분 의 어느 방향에 존재하는지 확인을 해서 두 선분이 교차하는지 확인할 수 있다. 각각 과 라하고 일 경우 교차가 일어났다고 할 수 있다.

이를 통해 생성된 맵은 Figure1과 같다.

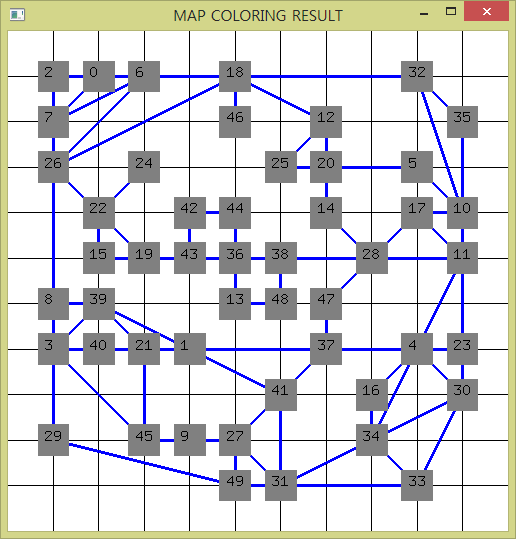


Figure1. Map Generation (,)

## Step 2. Start Map Coloring

### Step 2-1. Back Tracking only

주어진 Map Coloring 방법은 오직 Back Tracking만 사용하는 것과 Forward Checking, Arc Consistency를 이용하는 방법 3가지 있다. 기본적인 알고리즘은 주어진 Back Tracking알고리즘을 이용하였으며 그 과정을 아래와 같다.

1. 주어진 문제가 해결되었는가? 모든 Point에 Color가 배정되었다면 Success
2. 문제가 해결되지 않았다면 적합한 변수 을 선택한다. ()
3. 변수 의 값 중에서 적합한 값 를 선택한다. ()
4. 변수 에 값 를 배정하고, 위의 과정을 반복한다.

주어진 알고리즘 Reclusive Function 을 이용하여 구현되어 있다. 만약 실패하는 경우 기존에 배정된 색을 Remove한 뒤, 적합한 다른 값을 배정하는 것을 반복한다. 만약 모든 변수에 값을 배정할 수 없는 경우는 정해진 색의 수로 만들 수 있는 맵이 없는 경우이다.

기본적으로 3가지 색을 이용하여 구현하고, 만약 3가지 색을 이용해서 맵을 만드는데 실패했다면, 4가지 색을 이용하여 맵을 만든다.

적합한 변수를 선택하는 함수 은 MRV(Minimum Remaining Values)과 이 기법이 Tie-Breaker일 때 사용하는 MCV(Most Constraining Variable) 기법을 기반으로 변수를 선택한다. 그리고 값을 선택하는 함수 는 LCV(Least Constraining Value) 기법을 기반으로 값을 선택한다. 다만 Back Tracking만 사용하는 경우는 값을 RED부터 순차적으로 배정한다.

### Step 2-2. Back Tracking with Forward Check

Forward Checking 을 이용하는 경우 위에 기본 알고리즘을 기반으로 변수 에 값 를 배정하고 난 후 함수를 이용하여 변수 과 이웃한 변수들이 값 을 사용할 수 없도록 해야 한다. 이는 구조체 에 값을 추가하는 것으로 값의 사용 가능 여부를 저장했다.

만약 아직 값을 배정받지 못한 변수에 사용할 수 있는 값이 없다면, 즉 이웃하는 변수들이 각자 다른 를 배정받았다면, 해당 변수의 을 삭제하는 것뿐만 아니라 를 통해 삭제했던 이웃하는 변수들의 값을 복원할 필요가 있다. ()

### Step 2-3. Back Tracking with MAC (Maintaining Arc Consistency)

Arc Consistency를 이용하는 경우, 를 통해 변수를 선택한 후 해당 변수가 Consistency를 유지하고 있는지 확인을 한다. 이를 구현하는 함수는 이며, Line의 정보를 저장하고 있는 구조체 를 이용하여Queue를 만들어 사용하였다. 각 값들이 Consistent한지 확인하는 함수는 이며, 에 배정할 수 있는 색을 배정하였을 때 이웃하는 변수들에 배정할 수 있는 색이 남아있는지를 확인함으로써 Consistent를 확인한다. 만약 이웃 변수에 배정할 수 있는 색이 없는 경우 해당 변수의 값을 삭제하며, 해당 변수와 이웃하는 변수들의 Arc를 () 순으로 Queue에 다시 넣어 재확인한다.

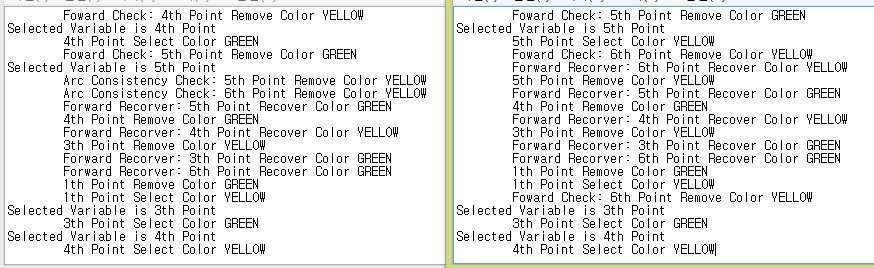


Figure2. Compare Forward Check and Arc Consistency

## Step 3. Show result on Window Screen. (Option)

main.cpp의 Boolean변수인 를 통해 기능을 키거나 끌 수 있다. MapScene.cpp에 구현되어 있으며, OpenGL을 이용하여 결과를 윈도우 창으로 띄우는 역할을 한다. 각 Point의 해당 좌표에 결과로 저장된 색의 사각형을 그리고 변수의 번호를 출력하며, 에 저장된 정보를 기반으로 선을 출력한다.

# Result and Comment

## 1. Option & Result

Boolean: [Log = OFF, Save = ON, ShowWindow = ON, AllEdge = ON]

Map Size:

Point Count:

기본적으로 Color의 수는 3이며 만약 맵 생성에 실패했을 경우, 4개의 Color를 이용하여 다시 시도한다. 실행 시간은 색 배정을 시작하기 전부터 색 배정이 끝났을 경우까지 이며, 3개의 색을 이용한 맵 생성에 실패했을 경우에 걸린 시간 또한 Time에 포함된다.

1. 3가지 색만을 이용하여 색 배정에 성공한 경우

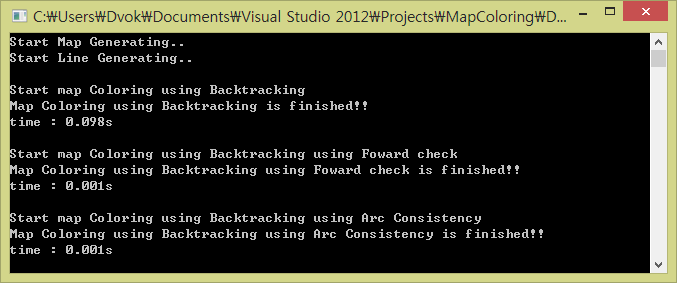


Figure3. 3-Coloring Map CONSOL

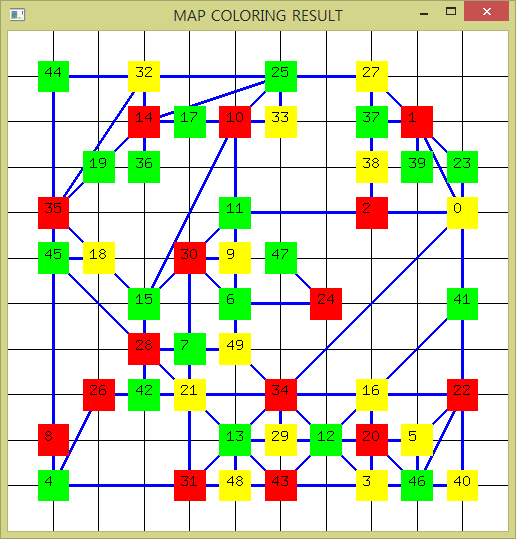


Figure4. 3-Coloring Map Window

1. 3가지 색을 이용하여 색 배정에 실패한 경우, 4가지 색을 이용한다.

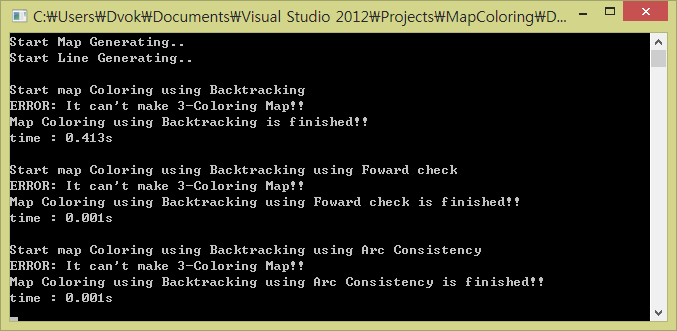


Figure5. 4-Coloring Map CONSOL

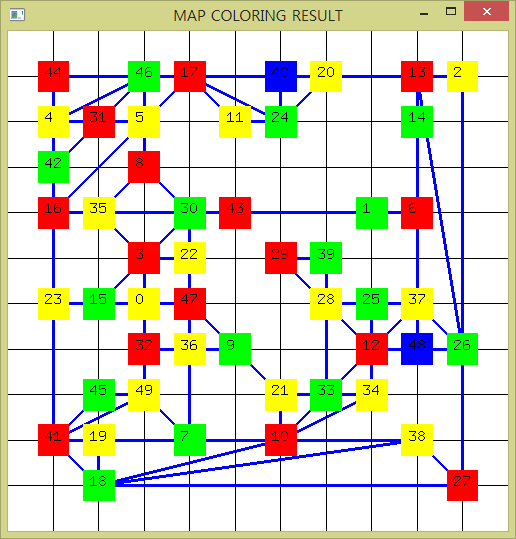


Figure6. 4-Coloring Map Window

## 2. Comment

오직 Back Tracking만 이용하는 경우, Breadth-First Search를 하여 Color를 배정하기 때문에 걸리는 시간이 다른 방법에 비해 오래 걸리는 것을 볼 수 있다. 반면 Forward Check나 Arc Consistency를 이용하면 실패를 빨리 알아 차리고 적합하지 않은 색에 대해 미리 삭제하기 때문에 속도가 훨씬 빠른 것을 볼 수 있었다. 이는 Point의 수가 많은 수록 그 차이가 심해지는데, 의 수가 90개로 설정하는 경우 오직 Back Tracking만 사용하는 경우 202초가 넘게 걸리는 반면, 다른 방법들은 0.002초로 이전과 큰 차이가 없는 것을 볼 수 있다.

또한 오직 Back Tracking만 이용하는 경우, 변수를 고르는 기법 중 MRV(Minimum Remaining Values)와 값을 고르는 기법인 LCV(Least Constraining Value)를 사용하지 않기 때문에 실행 시간이 크게 걸리는 것 같다.

마지막으로 Arc Consistency를 이용하는 경우, 가끔 Forward Check보다 시간이 오래 걸리는 경우가 있다. 이는 MRV(Minimum Remaining Values)기법을 이용해 변수를 선택하기 때문에 일어난다고 생각하는데, MRV 를 사용하는 경우 Figure7과 같이 변수로 뽑힌 의 색의 수가 의 색의 수보다 많은 경우가 나타나지 않는다. 오직 서로 색의 수가 같을 경우 Consistent한 지 판단하기 때문에 MRV 를 사용하는 경우 Arc Consistency의 효율이 좋지 않게 나오는 것 같다.

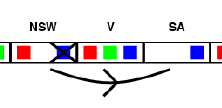
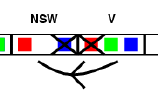
 

Figure7. ’s Values’s Values

# Future works (Remaining Problems)

## 1. Give it up too fast

가끔 Back Tracking만 이용하는 경우 3가지 색을 이용해서 맵을 생성하는데 성공했지만, 다른 경우 3가지 색을 이용해서 맵을 생성하는데 실패하는 경우가 있다. 또, Forward Check로도 3가지 색을 이용하는 경우 성공하는데, Arc Consistency를 이용하는 경우만 실패하는 경우도 있었다.

Color를 배정하고 실패할 때, 삭제했던 이웃 변수들의 색들을 복원하는 경우에서 문제가 있는지 모르겠다.

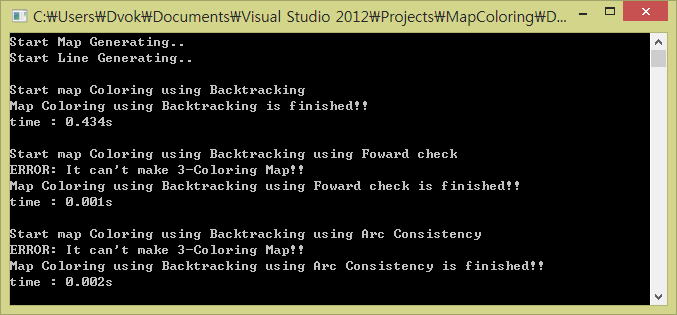
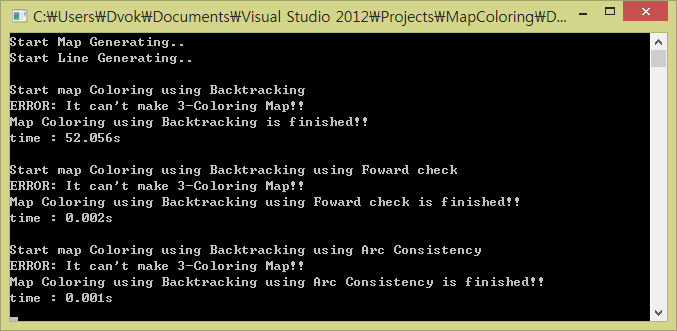


Figure8. Success only backtracking method

## 2. Use Too much Time

가끔 Back Tracking을 이용할 때 무지하게 오래 걸리는 경우가 있다. 보통 1초 미만에서 결과가 나오는데 가끔가다 Figure9과 같이 50초 이상 걸리는 경우가 있는데, 이 이유를 알 수가 없다. Log를 보려고 해도 Log File이 800MB가 넘어서 열리지도 않는다.



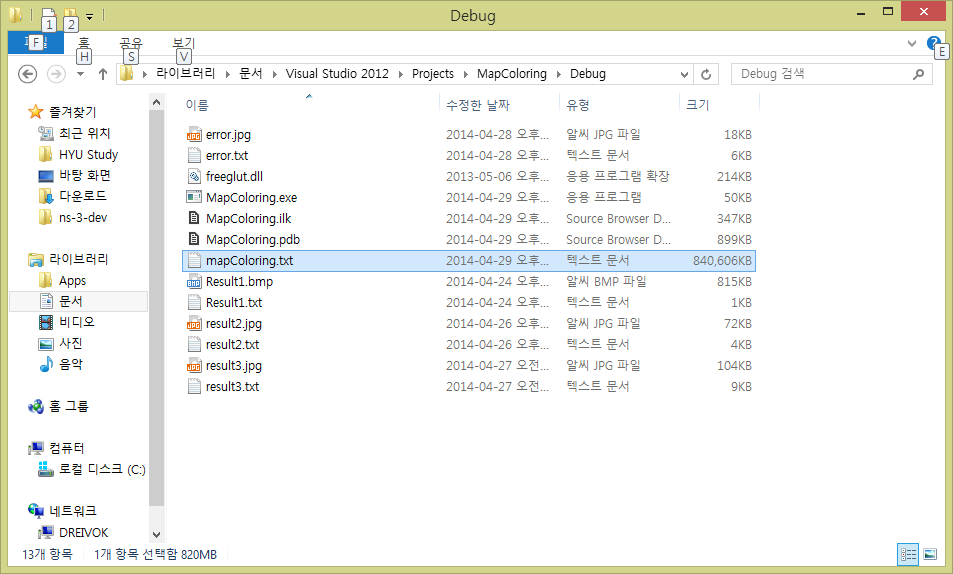


Figure9. Too much time in Backtracking