



**알고리즘설계 Project**

**“0-1 Knapsack 문제 분석을 위한**

**Dynamic Programming과 Genetic Algorithm의**

**구현 및 성능 비교, 분석”**

보고서 작성 서약서

1. 나는 타학생의 보고서를 베끼거나 여러 보고서의 내용을 짜집기하지 않겠습니다.

2. 나는 보고서의 주요 내용을 인터넷사이트 등을 통해 얻지 않겠습니다.

3. 나는 보고서의 내용을 조작하지 않겠습니다.

4. 나는 보고서 작성에 참고한 문헌의 출처를 밝히겠습니다.

5. 나는 나의 보고서를 제출 전에 타학생에게 보여주지 않겠습니다.

나는 보고서 작성시 윤리에 어긋난 행동을 하지 않고 정보통신공학인으로서 나의 명예를 지킬 것을 맹세합니다.

2016 년 6 월 19 일

학부 정보통신공학과

학년 3

성명 주예림

학번 12141768

# 개요

#### 알고리즘 설계 수업에서 배운 NP-COMPLETE 문제 중 0-1 KNAPSACK 문제를 DYNAMIC PROGRAMMING기법과 유전자 알고리즘(GENETIC ALGORITHM)을 이용하여 구현한 후 성능(수행시간)을 비교하고, 분석하였습니다. 그리고 그 문제를 해결하는 과정을 ANIMATION으로 보였습니다.

# 상세 설계 내용

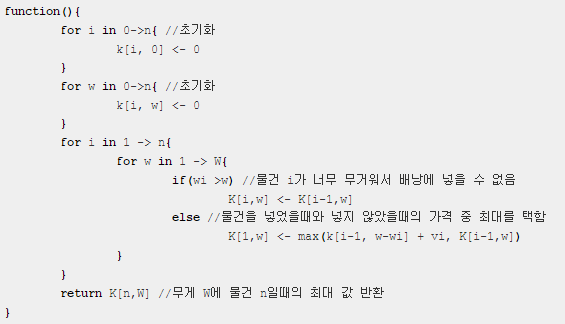
#### 대부분의 코드 해석은 주석에 설명하였습니다.

## 0) 0-1 KNAPSACK PROBLEM이란?

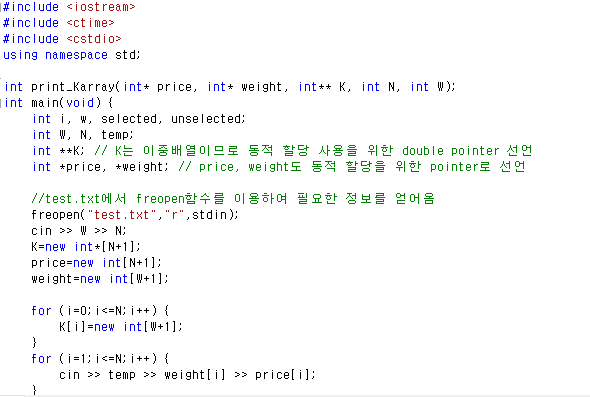
무게 W를 감당할 수 있는 배낭이 있을 때 N개 종류의 물건을 선택해서 넣을 수 있는데 각 물건은 무게와 가격이 각각 다릅니다. 최대 가격이 되기 위해 어떤 물건을 선택해야 하는지 알아내는 것이 Knapsack 문제입니다. Knapsack문제는 2가지가 있는데 하나는 0-1 Knapsack으로 물건을 자를 수가 없다는 것이고, 다른 하나는 Fraction Knapsack으로 물건을 자를 수 있다는 것입니다. 0-1 Knapsack 문제는 물건을 자를 수 없기 때문에 물건, 물건의 무게, 물건의 가격, 배낭의 남은 용량을 모두 고려해야 한다는 특징이 있습니다.

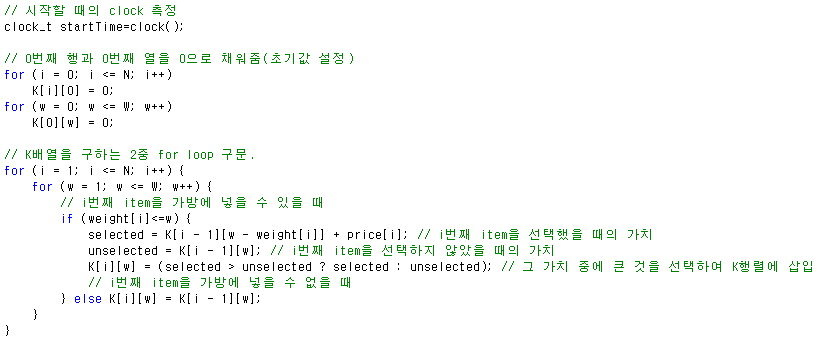
## dP기법을 이용한 문제 해결

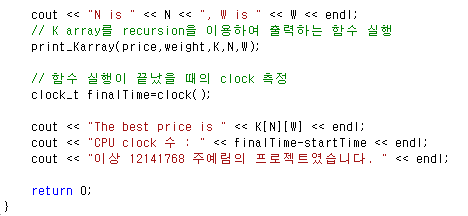
DP(Dynamic Programming)은 problem의 최적 해를 구하기 위해 sub problem(문제를 잘게 쪼갠 부분문제들)을 먼저 해결하여 배열에 저장한 뒤, 그 값들을 이용하여 큰 문제를 해결해 나간다는 특징이 있습니다. Pseudo code는 다음과 같습니다.



아래 코드는 위의 pseudo code를 참고하여 작성한 코드입니다. Solution은 source file인 DP\_Knapsack.cpp와 resource file인 test.txt로 이루어져 있습니다. Knapsack 함수는 따로 만들지 않았고 main함수에 넣었으며 K array를 출력할 때 recursion을 사용하기 위해 print\_Karray라는 함수를 선언하여 주었습니다.





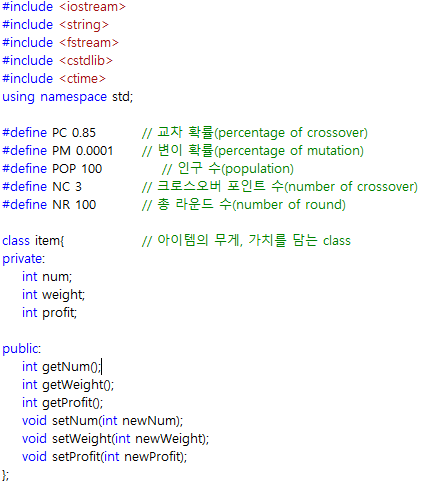


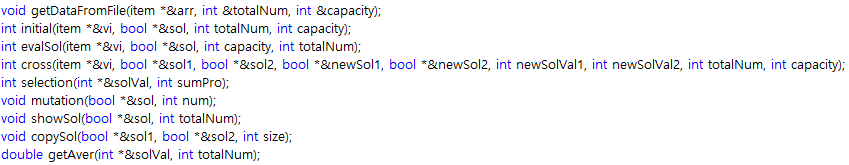
## GA기법을 이용한 문제 해결

GA(Genetic Algorithm)란, 어떤 문제를 컴퓨터 프로그램으로 해결함에 있어서 인류의 유전이라는 특성을 프로그램에 도입하여 좋은 답에 다가가도록 하는 알고리즘입니다. 유전 알고리즘은 명확한 해법이 없는 경우에도 해법을 생각하지 않고 유전자 염색체를 만든 다음 자식들끼리 세대를 거듭하고 교배하다 보면 자연스레 좋은 해에 도달할 수 있는 장점이 있습니다.

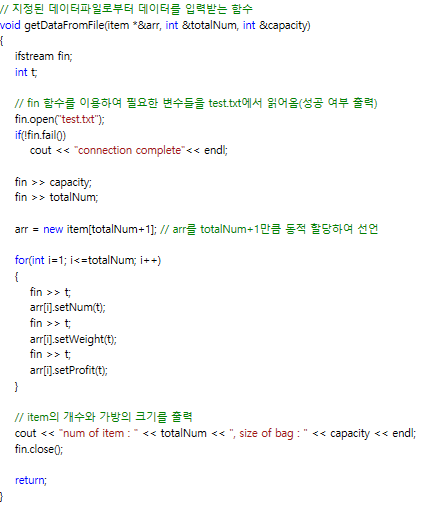
저는 knapsack에 최적화된 **encoding**을 고안하다가, 각 자리의 숫자는 해당 물건을 넣는지의 여부를 뜻하도록 인코딩을 하였습니다. 예를 들어, 4번째 숫자가 1이면 4번째 item이 들어간 경우이고, 10번째 숫자가 0이면 10번째 item이 들어가지 않은 경우라고 할 수 있습니다. 그리하여 염색체의 길이는 item의 개수와 같게 됩니다. **Selection**은 룰렛 선택 방법(Roulette whell selection)을 사용하였습니다. 각 염색체의 적합도에 비례하는 만큼 roulette의 영역을 할당한 다음(저는 확률의 개념으로 이용하였습니다.) 랜덤으로 룰렛을 돌려 화살표가 가리키는 영역의 염색체를 선택하는 방법입니다. 그리하여 적합도가 높은 것은 선택될 확률이 그만큼 많고, 적합도가 낮은 것은 선택될 확률이 상대적으로 낮아집니다. 즉 엘리트주의가 반영된 방법이라고 볼 수 있습니다. **Crossover**를 할 때에는 Single Point crossover를 사용하였습니다. 교차시킬 부분을 임의로 선택하여 그 이후의 해들을 서로 교차시키는 방법입니다. 저는 한번으로는 다양성 유지에 부족하다고 생각하여 NC(number of crossover)만큼 교차를 더 시키도록 하였습니다. 그리고 **Mutation**으로는 임의의 지점들을 선택하여 확률적으로 염색체의 유전자가 반전되도록 하였습니다. 그 외의 parameter로는 **세대교체** 시 부모 둘을 교배하여 자식을 낳았을 경우, 부모를 제거하고 그 자리에 자식을 배치하도록 하였습니다. 또한 **각 세대의 평균 가치와 최대 가치**를 출력해 줌으로써 세대를 거듭할수록 가치가 비슷하게 높다는 것을 관찰할 수 있도록 하였습니다. 하단은 앞서 설명한 기법들을 적용한 코드입니다. 헤더 파일 하나와 함수를 구현한 cpp파일, main 함수가 있는 파일 그리고 리소스 파일인 test.txt가 있습니다.

헤더 파일입니다. 교차 확률은 0.85, 변이 확률은 0.0001(0.1%), population은 100, 교차를 3번 하도록 하였고 세대는 총 100세대까지 생산하는 것으로 하였습니다.

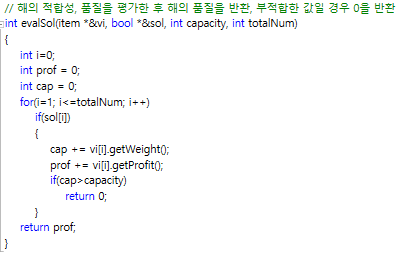


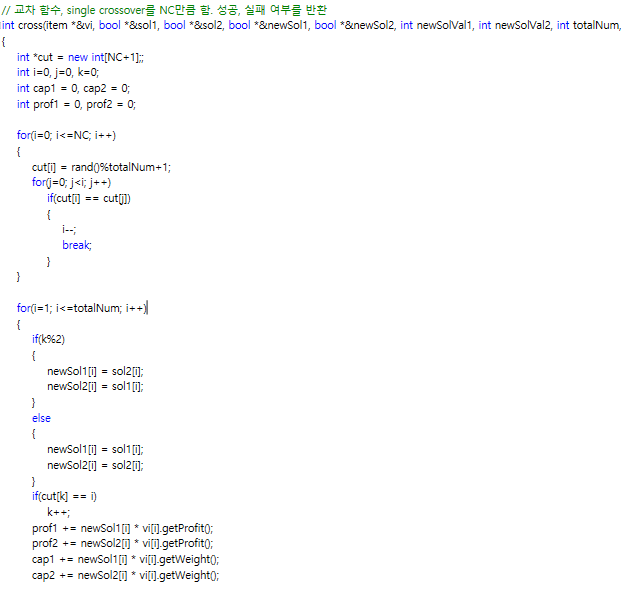


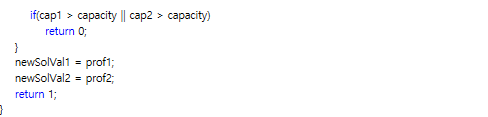
함수 구현부입니다. getDataFromFile 함수에서는 Fstream을 이용하여 test.txt파일을 읽어 와 item클래스의 배열에 할당이 가능하도록 구현하였습니다.



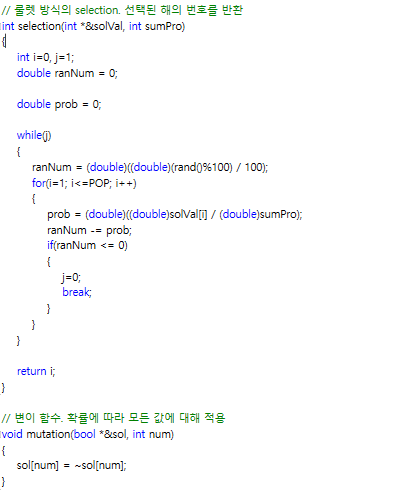
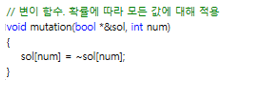
evalSol함수는 roulette wheel selection에 사용 될 적합성(fitness)을 구하기 위한 함수입니다.

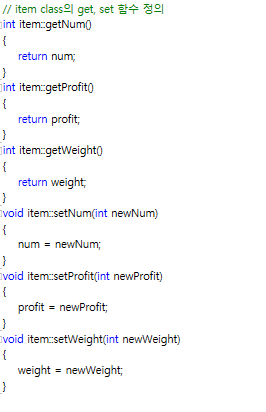
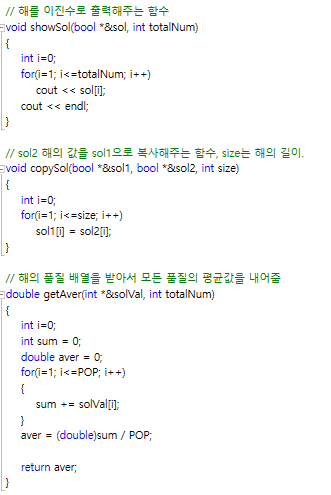






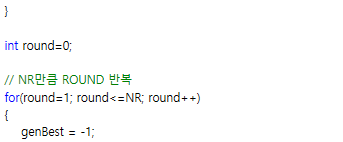
Selection과 mutation 선언부입니다. 또 기타 필요한 함수들도 선언해 주었습니다.

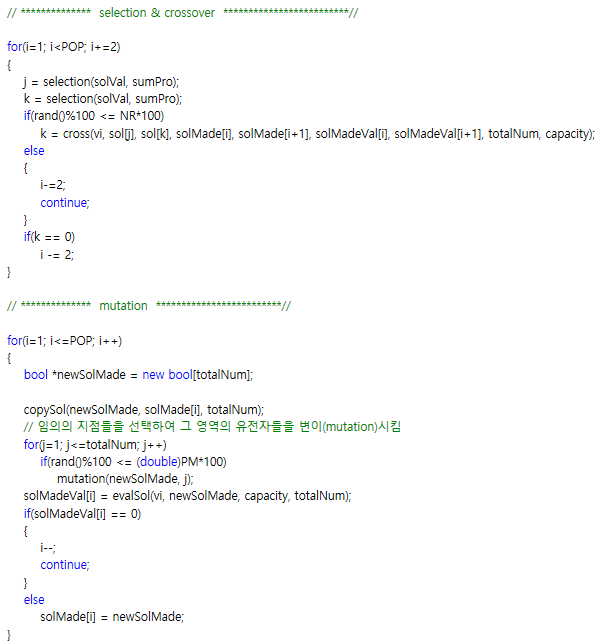


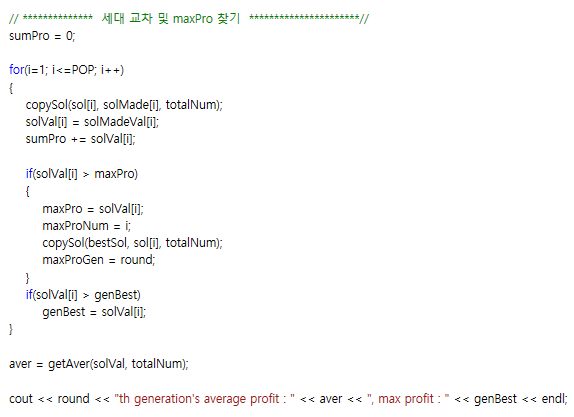


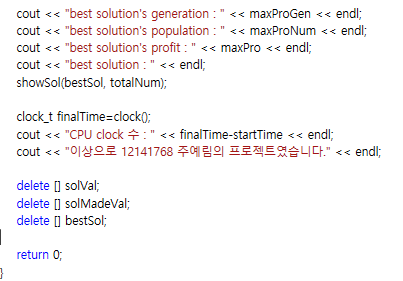
대망의 main 함수입니다.







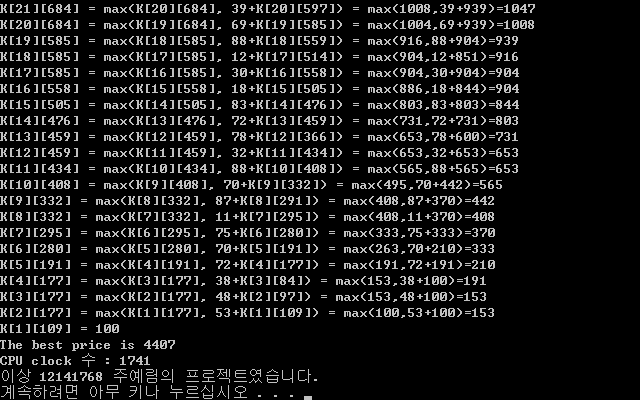


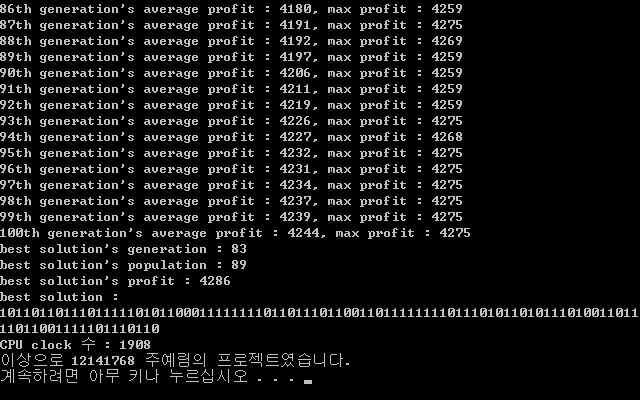


각 세대의 평균 가치, 최대 가치 그리고 전 세대를 통틀어 최고의 해가 있는 generation, population, profit 그리고 solution까지 출력하여 주었습니다. 또 성능 비교를 위한 CPU clock도 출력하도록 하였습니다.

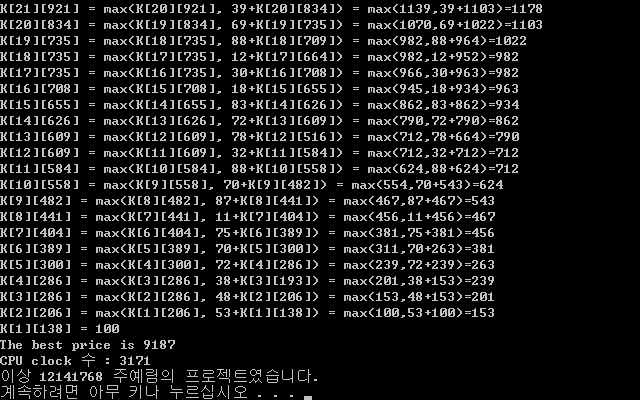
# 실행 화면

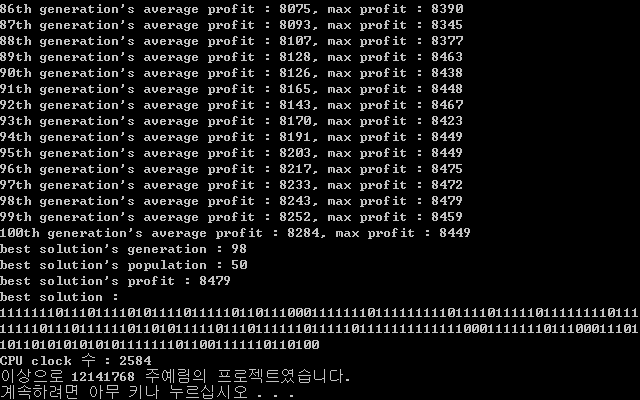
## Data set : item=100, size=3000





## Data set : item=200, size=7000





## Data set : item=300, size=10000





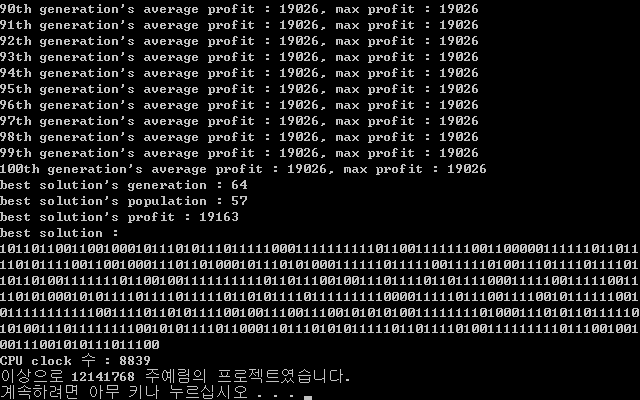
## Data set : item=400, size=14500





## Data set : item=500, size=17000





# 4. 그래프 및 분석

위의 data set을 표로 만들고, 꺾은선 그래프를 통하여 성능 및 실행시간을 비교분석하였다.

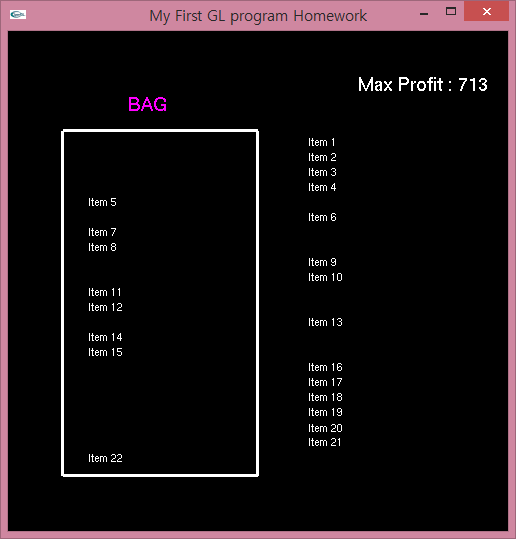


유전자 알고리즘과 동적 프로그래밍의 실행 시간을 분석해 보면, 동적 프로그래밍의 경우에는 n과 W의 관계는 무관하므로 W=n! 라고 한다면 수행 시간은 이 된다. 일일이 구하지 않고 필요한 경우만 계산하는 경우에는 최악의 경우, 시간에 수행될 수 있다. 유전자 알고리즘의 경우에는 이라고 할 수 있는데, 이 때 n은 item의 개수, m은 세대 수, l은 유전자의 수이다. 그러므로 m과 l이 작을 때는 선형 시간과 비슷해지는 반면, m과 l이 n에 가까워진다면 최악의 경우 을 기록할 수 있다. 이러한 전반적인 특징에서 그래프가 시사하는 바와 같이 성능에서는 동적 프로그래밍이 항상 최적 해를 찾으므로 우월할 수 밖에 없다. 하지만 실행 시간도 그만큼 오래 걸린다고 할 수 있다. 유전자 알고리즘은 guessing을 사용하므로 성능과 실행 시간이 들쑥날쑥 하다. 동적 프로그래밍과 비슷하거나, 나쁘거나 또는 좋거나 한 경우를 5개의 data set에서 모두 관찰할 수 있었다. 이러한 유전자 알고리즘의 특징은 더 좋은 기법들을 사용하거나 parameter의 조건(예를 들어, 교차율이나 유전 율 또는 세대 수 같은) 또는 종료조건 추가 여부에 따라 발전할 수 있을 것이다.

# 5. Animation 구현

Dynamic Programming을 OpenGL을 이용하여 구현하였습니다. Genetic Algorithm도 같은 방식으로 구현하면 됩니다. 코드는 zip파일에 첨부하였고 아래는 결과 창입니다.

N=22, W=300일 때



N=45, W=1500 일 때



