

Algorithm Project

**NP-Complete문제 중 0-1 Knapsack문제를**

**Dynamic Programming기법과 유전자 알고리즘(Genetic Algorithm)을**

**이용하여 구현한 후 성능(수행시간)을 비교 분석하라.**

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.

2018년 6 월 24 일

학부 정보통신공학과

학년 4

성명 유덕환

학번 12151584



1. ***개요***

NP-complete 문제 중 하나인 0-1 knapsack 문제를 DP(Dynamic programming)와 GA(Genetic Algorithm)으로 구현하여서 성능을 비교 분석한다.

* 어떤 인코딩 기법과 선택 연산자, 교배 연산자, 변이 연산자 등이 사용되었는지 그리고 어떻게 scaling을 했는지 자세히 서술한다.
* 그 외에 어떤 파라미터 값을 이용했는지 기술한다
* DP와 GA에 대해 5개 이상의 data set에 대하여 그래프를 그리고 성능을 비교 분석 한다. 즉, 최소한 그래프가 5개 이상이어야 한다.

1. ***구현***

**가.GA 기법**

|  |
| --- |
| **1)Encoding**  **- Binary 인코딩 기법**  TSP 처럼 순열 조합을 이용한 것이라면 실수 표현 기법등을 사용하였겠지만, 여기서는 i번째 아이템이 배낭에 들어갔는지 1, 들어가지 않았는지 0의 두가지 경우의 수밖에 없기 때문에 binary 인코딩 기법을 사용하였습니다.  **2)Selection :**  **적응도의 합 만큼의 int형 배열을 만들어서**  **각 개체의 적응도의 값 만큼의 개수에 해당 개체의 번호를 넣는다.**  Ex) 0번째 개체의 적응도가 10  1번째 개체의 적응도가 5이면  k번째 개체의 적응도가 6이면  Rulet[] = 0,0,0,0,0,0,0,0,0,0,1,1,1,1,1 .. .. k,k,k,k,k,k, .. ..  위와 같이 만들었다.      **실제 Roulette 배열의 출력.**  적응도의 합을 상한선으로 갖는 난수를 발생시키고, 해당 난수에 해당하는 rulet 배열의 값은 룰렛을 돌려서 나온 개체의 번호이다. |

|  |
| --- |
| **3)Mutation – IM** 사용    0번째를 3번째 위치에 삽입하였다.  과정은 2개의 난수를 p1과 p2에 저장 후, p1의 위치에 있는 것을 p2로 옮기는 것이다. 옮기기 위해서 p1+1부터 p2 까지는 Left Shift 해준다.    결과를 확인하고자 Test시 사용했던 Code. 이를 모든 things에 적용시킬 것. |

|  |
| --- |
| **4)Crossover**    1번째줄 : Parent 1,  2번째줄 : Parent 2  3번째줄 : baby 1  4번째줄 : baby 2  5번째줄 : baby 1과 baby 2의 Fitness    출력 시 사용한 코딩법.  방식 : baby1[k] = 50%확률로 Parent1[k] || Parent2[k]    효과 : Child의 염색체에는 Parent1과 Parent2의 유전자가 골고루 들어가게 된다.  추가 : Child를 만들고 만든 Child의 Fitness를 계산하여 Fitness==0일 경우 새로운 부모를 찾아서 다시 교배시킴.      5)Elitism    적응도를 기준으로 내림차순으로 정렬된 개체들의 앞쪽 10% 를 다음세대에 포함시키고, 나머지 0.9%는 룰렛 -> 교배 혹은 룰렛을 이용해 나온 개체들로 구성하였다.  6) Fitness    적응도를 구할 때에는 각 개체가 가지고 있는 Item들의 Price를 다 더한 값으로 해주었다. 무게도 같은 방식으로 다 더해주었는데, 만약 총 무게보다 넘어섰다면 적응도를 0으로 만들어주었다.  이 적응도에 특이한 Scaling은 하지 않았다. |

**나.GA 파라미터**

|  |
| --- |
|  |

**다. Code**

**1) KnapsackDP**

|  |
| --- |
| **import** java.util.Random;  //import asdf.thing;  **class** thing{  **public** thing(**int** w, **int** p) {  weight = w;  price = p;  }  **public** **int** weight;  **public** **int** price;  } // Item의 무게와 가치를 함께 담을 클래스.  **public** **class** Knapsack {  **private** **static** **int** *N* =100; // Item 개수 입력.  **private** **static** **int** *W* = 7000; // 담을수 있는 최대 무게 입력.  **private** **static** **int** *dp*[][] = **new** **int**[*N*+1][*W*+1];  // Reculsive Property를 적용할 2차원 배열 생성. 0행과 0열은 쓰지 않음.  **public** **static** **int** Max(**int** a, **int** b) {  **if**(a>=b) **return** a;  **else** **return** b;  }  **public** **static** **void** OptimalKnapsack(thing list[]) {  **for**(**int** i=1;i<=*N*;i++) {  **for**(**int** j=1;j<=*W*;j++) {  **if**(j < list[i].weight){  *dp*[i][j] = *dp*[i-1][j];  // Reculsive Property  }  **else** {  *dp*[i][j] = *Max*(*dp*[i-1][j],*dp*[i-1][j-list[i].weight] + list[i].price);  // Reculsive Property  }  //계속 두개중 하나로 분기를 하기 때문에 2^n의 복잡도를 가지게됨.  }  }  }  **public** **static** **void** print() {  **for**(**int** i=1;i<*N*+1;i++) {  **for**(**int** j=1;j<*W*+1;j++) {  System.***out***.print(*dp*[i][j] + " ");  }  System.***out***.println();  } // 제대로 들어가있는지 확인하기 위한 Code  //제출용에선 사용 X  }  **public** **static** **void** items(thing list[]) {  **int** w=*W*;  // 선택된 Item이 무엇이 있는지 확인해보기 위해서 만든 Code.  System.***out***.print("선택된 Items\n:");    **for**(**int** i=*N*;i>=1;i--) {  //System.out.println("["+(i)+"]"+"["+w+"]에 대해 보겠다.");  **if**(w-list[i].weight<0) **continue**;  //System.out.println("["+(i-1)+"]"+"["+w+"] = " + dp[i-1][w] + ", " + "["+(i-1)+"]"+"["+(w - list[i].weight) + "] + " + list[i].price + " = " + (dp[i-1][w-list[i].weight] + list[i].price));  **if**(*dp*[i-1][w] < *dp*[i-1][w-list[i].weight] + list[i].price) {  System.***out***.print(i + " ");  w = w - list[i].weight;  }  }  System.***out***.println();  }  **public** **static** **void** main(String args[]) {  System.***out***.println("12151584 유덕환 알고리즘 프로젝트");  thing [] list = **new** thing[*N*+1];    list[1] = **new** thing(29,22);  list[2] = **new** thing(56,89);  list[3] = **new** thing(125,97);  list[4] = **new** thing(43,38);  list[5] = **new** thing(26,24);  list[6] = **new** thing(57,50);  list[7] = **new** thing(117,95);  list[8] = **new** thing(65,77);  list[9] = **new** thing(45,79);  list[10] = **new** thing(15,27);  list[11] = **new** thing(51,63);  list[12] = **new** thing(37,36);  list[13] = **new** thing(23,33);  list[14] = **new** thing(33,27);  list[15] = **new** thing(66,71);  list[16] = **new** thing(68,67);  list[17] = **new** thing(80,80);  list[18] = **new** thing(44,61);  list[19] = **new** thing(144,99);  list[20] = **new** thing(38,46);  list[21] = **new** thing(104,92);  list[22] = **new** thing(27,26);  list[23] = **new** thing(36,38);  list[24] = **new** thing(58,44);  list[25] = **new** thing(75,63);  list[26] = **new** thing(21,20);  list[27] = **new** thing(42,46);  list[28] = **new** thing(81,71);  list[29] = **new** thing(94,75);  list[30] = **new** thing(24,17);  list[31] = **new** thing(29,56);  list[32] = **new** thing(30,27);  list[33] = **new** thing(25,36);  list[34] = **new** thing(48,54);  list[35] = **new** thing(120,96);  list[36] = **new** thing(34,56);  list[37] = **new** thing(152,104);  list[38] = **new** thing(10,11);  list[39] = **new** thing(9,10);  list[40] = **new** thing(48,36);  list[41] = **new** thing(51,100);  list[42] = **new** thing(72,53);  list[43] = **new** thing(36,25);  list[44] = **new** thing(55,51);  list[45] = **new** thing(40,69);  list[46] = **new** thing(61,45);  list[47] = **new** thing(69,55);  list[48] = **new** thing(32,31);  list[49] = **new** thing(122,94);  list[50] = **new** thing(56,57);  list[51] = **new** thing(71,102);  list[52] = **new** thing(80,73);  list[53] = **new** thing(61,88);  list[54] = **new** thing(64,50);  list[55] = **new** thing(18,29);  list[56] = **new** thing(58,101);  list[57] = **new** thing(67,103);  list[58] = **new** thing(43,40);  list[59] = **new** thing(77,104);  list[60] = **new** thing(34,33);  list[61] = **new** thing(127,93);  list[62] = **new** thing(109,95);  list[63] = **new** thing(75,69);  list[64] = **new** thing(66,85);  list[65] = **new** thing(8,9);  list[66] = **new** thing(102,104);  list[67] = **new** thing(73,63);  list[68] = **new** thing(25,43);  list[69] = **new** thing(9,14);  list[70] = **new** thing(121,94);  list[71] = **new** thing(80,104);  list[72] = **new** thing(19,16);  list[73] = **new** thing(53,91);  list[74] = **new** thing(94,64);  list[75] = **new** thing(109,82);  list[76] = **new** thing(10,8);  list[77] = **new** thing(14,12);  list[78] = **new** thing(10,9);  list[79] = **new** thing(27,44);  list[80] = **new** thing(88,90);  list[81] = **new** thing(7,8);  list[82] = **new** thing(31,57);  list[83] = **new** thing(90,61);  list[84] = **new** thing(7,7);  list[85] = **new** thing(9,12);  list[86] = **new** thing(86,77);  list[87] = **new** thing(21,30);  list[88] = **new** thing(137,99);  list[89] = **new** thing(60,60);  list[90] = **new** thing(6,8);  list[91] = **new** thing(18,13);  list[92] = **new** thing(55,66);  list[93] = **new** thing(65,60);  list[94] = **new** thing(55,73);  list[95] = **new** thing(40,61);  list[96] = **new** thing(79,89);  list[97] = **new** thing(51,39);  list[98] = **new** thing(33,39);  list[99] = **new** thing(14,17);  list[100] = **new** thing(45,37);  **long** start = System.*nanoTime*();  *OptimalKnapsack*(list);  **long** end = System.*nanoTime*();  **long** elapsed = end - start;  System.***out***.println("최적해 : " + *dp*[*N*][*W*]);  System.***out***.println( "경과 시간 : " + ((**double**)elapsed / 1000000.0) + "ms");  //print();  *items*(list);    }  } |

**2) KnapsackGA**

|  |
| --- |
| **package** asdf;  **import** java.lang.reflect.Member;  **import** java.util.Arrays;  **import** java.util.Comparator;  **import** java.util.Random;  **class** thing{  **public** thing(**int** w, **int** p) {  weight = w;  price = p;  }  **public** **int** weight;  **public** **int** price;  }  **class** fit{ // Elitism을 보이기 위해 적응도 순으로 정렬을 위해 만든 Class.  **public** fit(**int** o, **int** p) {  order = o;  price = p;  }  **public** **int** order=0;  **public** **int** price=0;  }  **public** **class** KnapsackGA {  **private** **static** **int** *N* = 100; // Item 개수  **private** **static** **int** *Max\_W* = 7000; // 최대 담을수 있는 무게.  **private** **static** **int** *POPULATION* = 1000; // 집단 크기.  **private** **static** **int** *LENGTH* = *N*; //염색체 길이는 Item의 개수와 같다.  **private** **static** **int** *GENERATION* = 1000; // 세대 수  **private** **static** **double** *CROSSOVER* = 0.7; // 교배율  **private** **static** **double** *MUTATION* = 0.01; // 변이율  **private** **static** **double** *DPopt\_sol* = 5564; // DP에서 구한 최적해. 이에 도달하면 세대를 멈추도록 하였음.  **private** **static** **int**[][] *things* = **new** **int**[*POPULATION*][*LENGTH*]; // 집단이 들어가있는 공간.  **private** **static** thing[] *list* = **new** thing[*N*]; // Item을 담아둘 공간.  **private** **static** fit[] *Fitness* = **new** fit[*POPULATION*]; // 적응도의 계산을 담아둘 공간.  **private** **static** Random *rnd* = **new** Random();    **public** **static** **void** init() {    **for**(**int** i=0;i<*POPULATION*;i++) {  **for**(**int** j=0;j<*LENGTH*;j++) {  **if** (Math.*abs*(*rnd*.nextDouble())%1 >= 0.5) *things*[i][j]=1;  **else** *things*[i][j]=0;  // Binary로 Encoding. 1이면 해당 물건이 들어간것, 0이면 안들어간것.  // 50%으로 1 혹은 0이 결정된다.  }  }  *CalculateFit*();  }// make population  **public** **static** **void** CalculateFit() {  **for**(**int** k=0;k<*POPULATION*;k++) {  **int** Value =0;  **int** Weight =0;  **for**(**int** i=0;i<*LENGTH*;i++) {  Value += *things*[k][i]\**list*[i].price;  Weight += *things*[k][i]\**list*[i].weight;  // Binary Encoding 이므로 0 혹은 1이다.  // 1로 된 인덱스는 물건의 가치값을 곱하여 더한다.  // if(thing[k][i] == 1) 대신 줄여서 사용한것.  }  **if**(Weight > *Max\_W*) Value=0;  // 총 무게보다 넘어선 물건은 적응도를 0으로 만든다.  *Fitness*[k] = **new** fit(k,Value);  }  }// calculate fitness  **public** **static** **int** CalculateFit(**int**[] n) {  // 하나의 개체에 관한 적응도를 계산하는 함수.  // 교배시 자식의 적응도가 0인 경우에 대비하기 위해 만듬.  **int** Value =0;  **int** Weight =0;  **for**(**int** i=0;i<*LENGTH*;i++) {  Value += n[i]\**list*[i].price;  Weight += n[i]\**list*[i].weight;  }  **if**(Weight > *Max\_W*) Value=0;  **return** Value;  }// calculate fitness  **public** **static** **int** BestThing() {  // 집단 중에서 가장 최적의 해를 출력.  **int** best = 0;  **for**(**int** i=0;i<*POPULATION*;i++) {  **if** (*Fitness*[i].price>*Fitness*[best].price) best = i;  }  **return** best;  }  **public** **static** **int** FitValueSum() {  // Roulette을 만들기 위해서 필요한 적응도의 총 합.  **int** sum=0;  **for**(**int** i=0;i<*POPULATION*;i++) {  sum+=*Fitness*[i].price;  }  **return** sum;  }  **public** **static** **int**[] Mix(**int**[] parent1, **int**[] parent2) {  // 부모 개체 2개를 입력받아 하나의 자식을 내보낸다.  // 방식은 50% 확률로 특정 Index의 값은 Parent1의 값이고,  // 나머지 50% 확률로 Parent2의 값으로 만들어서, 자식은 2개의 부모 개체로부터 영향을 받음을 표현.    **int**[] child = **new** **int**[*LENGTH*];  **for**(**int** i=0;i<*LENGTH*;i++) {  **if**( Math.*abs*(*rnd*.nextDouble()) % 1 < 0.5) child[i] = parent1[i];  **else** child[i] = parent2[i];  }    **return** child;  }  **public** **static** **int**[][] twoChoros(**int**[] rulet,**int** sum) {  // 룰렛을 돌려서 두개의 염색체 개체를 반환하는 함수.  **int** a=rulet[Math.*abs*(*rnd*.nextInt()) % sum];  **int** b=a;  **while**(a==b) b=rulet[Math.*abs*(*rnd*.nextInt()) % sum];  **int**[][] tmp = **new** **int**[2][*LENGTH*];  tmp[0] = *things*[a];  tmp[1] = *things*[b];  **return** tmp;  }  **public** **static** **void** crossover(**int**[] rulet,**int** sum) {  // 교배함수.  **int**[][] newThings= **new** **int**[*POPULATION*][*LENGTH*];  **int** index=0;  **int**[] baby1 = **new** **int**[*LENGTH*];  **int**[] baby2 = **new** **int**[*LENGTH*];  fit[] tmp = *Fitness*;    Arrays.*sort*(tmp,**new** Comparator<fit>(){  @Override  **public** **int** compare(fit o1, fit o2){  **if**(o1 == **null** || o2 == **null** ) **return** 0;//null있을경우 리턴값 0 넘기기 null오류를 피하기 위함  **return** o2.price - o1.price;//숫자 내림차순  }  });    **for**(**int** i=0;i<*POPULATION*\*0.1;i++) {  newThings[index++] = *things*[tmp[i].order];  } // Elitism을 표현하기 위해서 적응도가 큰 순서대로 들어가있는 tmp 배열을 이용.  //앞쪽 10%을 다음 세대에 포함시킴.  **for**(**int** i=0;i<*POPULATION*\*0.45;i++) {      **int** [][] temp = *twoChoros*(rulet,sum);  **int**[] Parent1 = temp[0];  **int**[] Parent2 = temp[1];        **if** ( Math.*abs*(*rnd*.nextDouble())%1 < *CROSSOVER* ) {  baby1 = *Mix*(Parent1,Parent2);  baby2 = *Mix*(Parent2,Parent1);    **while**(*CalculateFit*(baby1)==0 && *CalculateFit*(baby2)==0) {  // 두 자식 모두 적응도가 0이라면 다시 부모를 뽑아서 교배를 다시 시킴.  temp = *twoChoros*(rulet,sum);  Parent1 = temp[0];  Parent2 = temp[1];  baby1 = *Mix*(Parent1,Parent2);  baby2 = *Mix*(Parent2,Parent1);  }    newThings[index++] = baby1;  newThings[index++] = baby2;  }**else** {  newThings[index++] = Parent1;  newThings[index++] = Parent2;  }  }  // newThings라는 곳에 POPULATION 수만큼의 개체를 집어넣은 후 이를 본래의 things로 대입.  *things* = newThings;  *CalculateFit*();  }// crossover  **public** **static** **void** mutation() { // 변이함수. IM을 사용하였다.  **for**(**int** i=0;i<*POPULATION*;i++) {  **if** ( Math.*abs*(*rnd*.nextDouble())%1 < *MUTATION* ) {  **int** p1=10,p2=10;  **while**(p1>=9) p1= Math.*abs*(*rnd*.nextInt()%10);  p2=p1;  **while**(p1==p2 || p1>p2) p2 = Math.*abs*(*rnd*.nextInt()%10);    **int** temp = *things*[i][p1];    **for**(**int** k=p1+1;k<=p2;k++)  *things*[i][k-1] = *things*[i][k];    *things*[i][p2]=temp;  }  }  *CalculateFit*();  }// mutation  **public** **static** **void** GA\_Knapsack() {  // 알고리즘을 총괄하는 함수.  *init*(); // 초기화.  **int** i=0; // 몇번째 세대에서 찾았는지 기록함.  **int** Best=0; // 최적해가 무엇인지 기록.  **for**(i=0;i<*GENERATION*;i++) {  Best = *Fitness*[*BestThing*()].price;  **if**( Best == *DPopt\_sol*) {  System.***out***.println("최적해 : " + Best + " Generation : " + i + "/" + *GENERATION*);  **break**;  }  **int** sum=*FitValueSum*();  **int**[] rulet = **new** **int**[sum];    **int** index=0;  **for**(**int** k=0;k<*POPULATION*;k++) {  **for**(**int** j=0;j<*Fitness*[k].price;j++) {  rulet[index++] = k;  }  }  *crossover*(rulet,sum);  //교배  *mutation*();  //변이  }  System.***out***.println("최적해 : " + Best + " Generation : " + i + "/" + *GENERATION*);  }  **public** **static** **void** main(String args[]) {  System.***out***.println("12151584 유덕환 알고리즘 프로젝트");    *list*[0] = **new** thing(45,37);  *list*[1] = **new** thing(29,22);  *list*[2] = **new** thing(56,89);  *list*[3] = **new** thing(125,97);  *list*[4] = **new** thing(43,38);  *list*[5] = **new** thing(26,24);  *list*[6] = **new** thing(57,50);  *list*[7] = **new** thing(117,95);  *list*[8] = **new** thing(65,77);  *list*[9] = **new** thing(45,79);  *list*[10] = **new** thing(15,27);  *list*[11] = **new** thing(51,63);  *list*[12] = **new** thing(37,36);  *list*[13] = **new** thing(23,33);  *list*[14] = **new** thing(33,27);  *list*[15] = **new** thing(66,71);  *list*[16] = **new** thing(68,67);  *list*[17] = **new** thing(80,80);  *list*[18] = **new** thing(44,61);  *list*[19] = **new** thing(144,99);  *list*[20] = **new** thing(38,46);  *list*[21] = **new** thing(104,92);  *list*[22] = **new** thing(27,26);  *list*[23] = **new** thing(36,38);  *list*[24] = **new** thing(58,44);  *list*[25] = **new** thing(75,63);  *list*[26] = **new** thing(21,20);  *list*[27] = **new** thing(42,46);  *list*[28] = **new** thing(81,71);  *list*[29] = **new** thing(94,75);  *list*[30] = **new** thing(24,17);  *list*[31] = **new** thing(29,56);  *list*[32] = **new** thing(30,27);  *list*[33] = **new** thing(25,36);  *list*[34] = **new** thing(48,54);  *list*[35] = **new** thing(120,96);  *list*[36] = **new** thing(34,56);  *list*[37] = **new** thing(152,104);  *list*[38] = **new** thing(10,11);  *list*[39] = **new** thing(9,10);  *list*[40] = **new** thing(48,36);  *list*[41] = **new** thing(51,100);  *list*[42] = **new** thing(72,53);  *list*[43] = **new** thing(36,25);  *list*[44] = **new** thing(55,51);  *list*[45] = **new** thing(40,69);  *list*[46] = **new** thing(61,45);  *list*[47] = **new** thing(69,55);  *list*[48] = **new** thing(32,31);  *list*[49] = **new** thing(122,94);  *list*[50] = **new** thing(56,57);  *list*[51] = **new** thing(71,102);  *list*[52] = **new** thing(80,73);  *list*[53] = **new** thing(61,88);  *list*[54] = **new** thing(64,50);  *list*[55] = **new** thing(18,29);  *list*[56] = **new** thing(58,101);  *list*[57] = **new** thing(67,103);  *list*[58] = **new** thing(43,40);  *list*[59] = **new** thing(77,104);  *list*[60] = **new** thing(34,33);  *list*[61] = **new** thing(127,93);  *list*[62] = **new** thing(109,95);  *list*[63] = **new** thing(75,69);  *list*[64] = **new** thing(66,85);  *list*[65] = **new** thing(8,9);  *list*[66] = **new** thing(102,104);  *list*[67] = **new** thing(73,63);  *list*[68] = **new** thing(25,43);  *list*[69] = **new** thing(9,14);  *list*[70] = **new** thing(121,94);  *list*[71] = **new** thing(80,104);  *list*[72] = **new** thing(19,16);  *list*[73] = **new** thing(53,91);  *list*[74] = **new** thing(94,64);  *list*[75] = **new** thing(109,82);  *list*[76] = **new** thing(10,8);  *list*[77] = **new** thing(14,12);  *list*[78] = **new** thing(10,9);  *list*[79] = **new** thing(27,44);  *list*[80] = **new** thing(88,90);  *list*[81] = **new** thing(7,8);  *list*[82] = **new** thing(31,57);  *list*[83] = **new** thing(90,61);  *list*[84] = **new** thing(7,7);  *list*[85] = **new** thing(9,12);  *list*[86] = **new** thing(86,77);  *list*[87] = **new** thing(21,30);  *list*[88] = **new** thing(137,99);  *list*[89] = **new** thing(60,60);  *list*[90] = **new** thing(6,8);  *list*[91] = **new** thing(18,13);  *list*[92] = **new** thing(55,66);  *list*[93] = **new** thing(65,60);  *list*[94] = **new** thing(55,73);  *list*[95] = **new** thing(40,61);  *list*[96] = **new** thing(79,89);  *list*[97] = **new** thing(51,39);  *list*[98] = **new** thing(33,39);  *list*[99] = **new** thing(14,17);  **long** start = System.*nanoTime*();  *GA\_Knapsack*();  **long** end = System.*nanoTime*();  **long** elapsed = end - start;    System.***out***.println( "경과 시간 : " + ((**double**)elapsed / 1000000.0) + "ms");    }    } |

1. ***실행 결과 및 그래프 비교***

**가. DP실행결과 예시**

|  |
| --- |
|  |

1. **GA실행결과 예시**

|  |
| --- |
|  |

Data Set이 커짐에 따라 출력하는 시간이 크므로 최적해에 대해서만 출력하겠습니다.

**다.Data Set에 따른 DP 와 GA 비교**

**1) # Item 10**

|  |
| --- |
| DP    GA  Setting |

|  |
| --- |
| DP Result  GA Result |

1. **# Item 30**

|  |
| --- |
| DP    GA  Setting |

|  |
| --- |
| DP Result  GA Result |

1. **# Item 50**

|  |
| --- |
| DP    GA  Setting |

|  |
| --- |
| DP Result    GA Result |

1. **# Item 70**

|  |
| --- |
| DP    GA  Setting |

|  |
| --- |
| DP Result    GA Result |

1. **# Item 100**

|  |
| --- |
| DP    GA  List는 따로 메모장으로 첨부. |

|  |
| --- |
|  |

**6) DP, GA비교 그래프**

***4.고찰***

이번 과제는 NP 문제중 하나인 Knapsack Problem 문제를 DP방식과 GA 방식으로 구현, 성능 비교를 하는 과제다.

O(2^n)의 시간복잡도를 가지기 때문에 다항시간안에 결정할 수 없는 알고리즘이다. 결과적으로 Item의 개수가 10, 30, 50, 7, 100의 GA의 입장에서 보자면 적은 수의 Item에 대한 Test를 했기 때문인지는 모르겠으나, 성능면에선 DP를 이용한 Knapsack이 GA를 이용한 knapsack보다 성능이 뛰어났다.

본인은 GA를 구성함에 있어서,

* Binary Encoding(해당 Item의 유무)
* 100 ~ 1000 Populations
* 100 ~ 1000 Generation
* 0.7 Crossover rate
* 0.05 Mutation rate
* 0.1 Elitism rate
* Roulette
* Fitness : 가지고 있는 Item들의 Price를 더한값.
* Insertion Mutation
* Crossover는 위의 설명. 기존 기법을 사용하지 않음.

을 사용하였다. 강의노트에 나와있는 PMX, OBX 등의 기법을 이용해서 Crossover를 시도해보았지만, 최적해에 근접하지도 못하는 값에 수렴해버려서 Binary Encoding의 경우에는 사용하기에 적절하지 못하다는 판단을 내렸다.

이번 과제를 통해 GA를 맨바닥에서부터 시작하여 결과를 도출해내는 완성하기 까지를 두두세번 거친 것 같다. GA의 장점인 알고리즘을 알지 못해도 최적해 혹은 최적해에 근접한 해를 찾을 수 있다는 장점을 이용해서 다른 문제 해결에 응용하겠다.