C2C Caleb Song

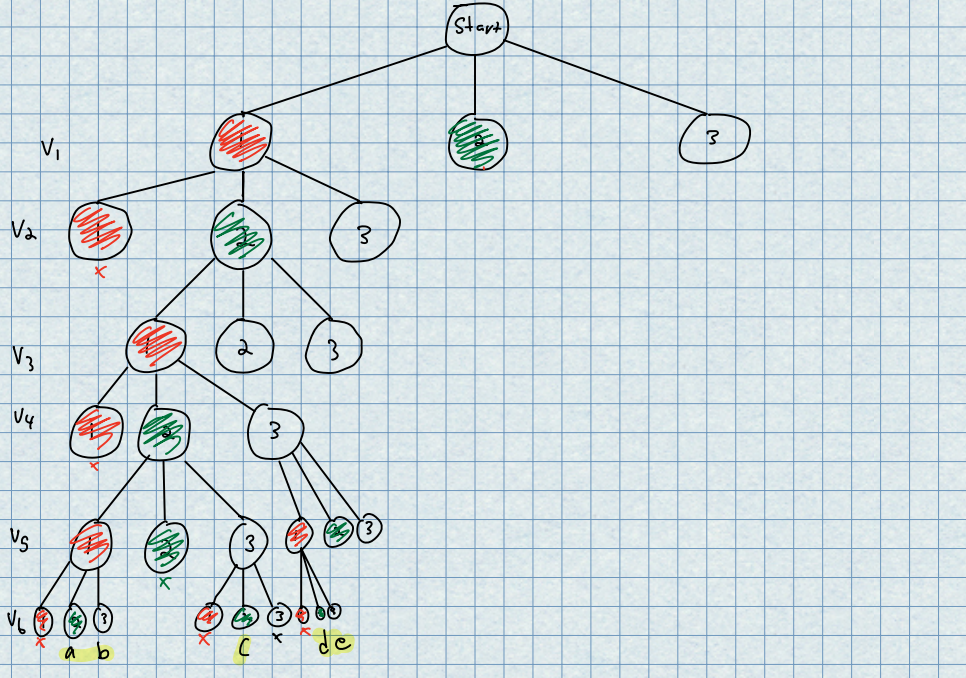
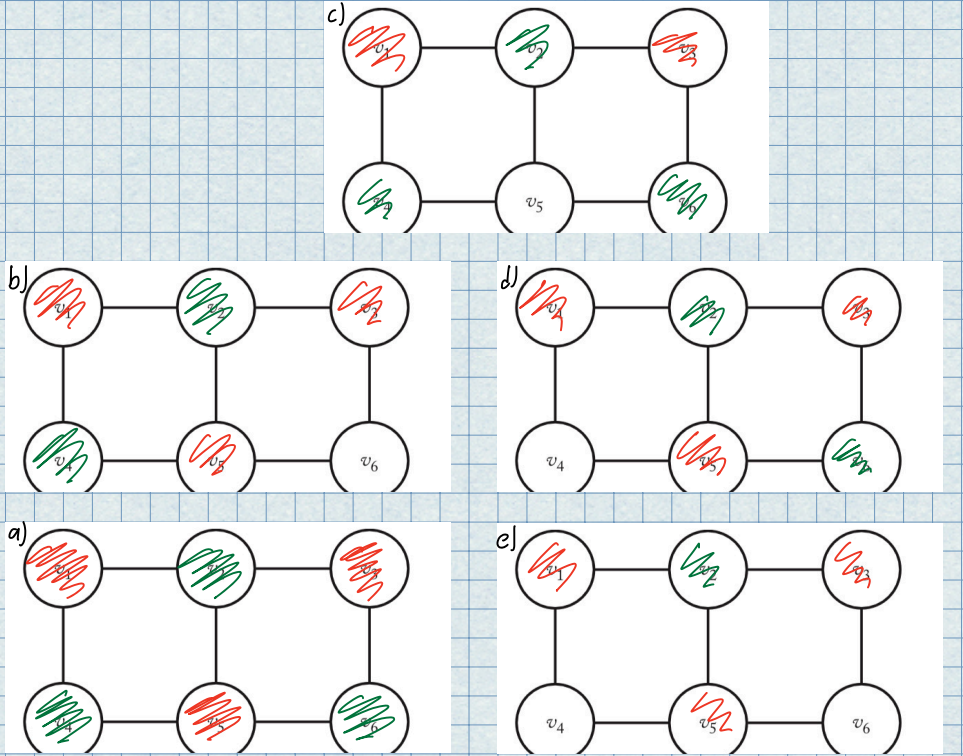
Documentation: EI with Dr. Hadfield 4/2.

Question 1:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Number of solutions | Number of promising nodes | Number of all nodes | Ratio of promising to number of all nodes | Run-time in nanoseconds |
| N=4 | 2 | 17 | 341 | 0.04985337243401759 | 11000.0 |
| N=6 | 4 | 153 | 55987 | 0.0027327772518620395 | 263000.0 |
| N=8 | 92 | 2057 | 1.9173961E7 | 1.0728091081441127E-4 | 1216800.0 |
| N=10 | 724 | 35539 | 1.1111111111E10 | 3.198510000031985E-6 | 9144100.0 |
| N=12 | 14200 | 856189 | 9.726655034461E12 | 8.802501959477023E-8 | 1.522105E8 |

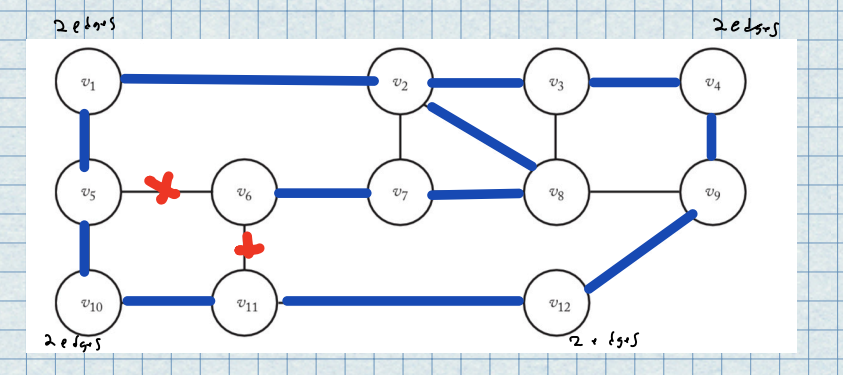
import java.lang.Math;  
public class Main {  
 public static int[] *col*; //what column the queen is on in the row  
 public static int *n*; //number of queens  
 public static long *counter*; //count number of solutions  
 public static long *promisingCounter*; //count number of promising nodes  
 public static double *numberNodes*; //count number of all nodes  
 public static double *ratio*; //ratio of promising to total nodes  
 public static void main(String[] args) {  
 *n* =12;//set n based on how many queens are desired there are  
 *col* = new int [*n*+1];//1-based indexing so initialize columns array size N+1  
 *counter* = 0;//set number of solutions to 0  
 *promisingCounter* = 0;//set promising counter to 0  
 *numberNodes* = (Math.*pow*(*n*,(*n*+1))-1)/(*n*-1); //using formula from page 213 calculates total number nodes  
 long startTime; //start time to track runtime  
 double actualTime; //end time to track runtime  
 startTime = System.*nanoTime*();//start time  
 *queens*(0);//run queens with i = 0  
 actualTime = (System.*nanoTime*() - startTime); //calculate runtime do not divide by 1.0E09 to keep it in nanoseconds  
 *ratio* = (double) *promisingCounter* /*numberNodes*; // find ratio  
 //Print out desired results  
 System.*out*.println("Number of solutions found: " + *counter*);  
 System.*out*.println("Number of promising nodes: " + *promisingCounter*);  
 System.*out*.println("Number of total nodes: " + *numberNodes*);  
 System.*out*.println("Ratio of promising to all nodes: " + *ratio*);  
 System.*out*.println("Runtime in nanoseconds: " + actualTime);  
 }  
 public static void queens(int i){  
 int j;  
 //check to see if next placement on board is promising  
 if(*promising*(i)){  
 //if i == n then reached end of the board with a valid solution  
 if(i == *n*){  
 //uncomment out the commented lines to print the solutions  
// for(int t = 1; t <=n;t++){  
// System.out.print(col[t] + " ");  
// }  
 *counter*+=1; // add to the counter if board reached with valid solution  
// System.out.println();  
 }  
 else{  
 //else keep going through board  
 for (j = 1; j<=*n*;j++){  
 //place queen on board  
 *col*[i+1] = j;  
 //recursively run queens at next column  
 *queens*(i+1);  
 }  
 }  
 }  
 }  
 public static boolean promising(int i){  
 //set condition to true  
 boolean condition = true;  
 //set k = 1 and then iterate through board columns  
 int k = 1;  
 while(k < i && condition){  
 //check to see if queens threaten other queens when place on board  
 if (*col*[i] == *col*[k] || Math.*abs*(*col*[i]-*col*[k]) == i-k){  
 condition = false;  
 }  
 k++;  
 }  
 //if passes above loop then add to promising counter  
 if (condition){  
 *promisingCounter*+=1;  
 }  
 //return condition either true or false  
 return condition;  
 }  
}

Question 2

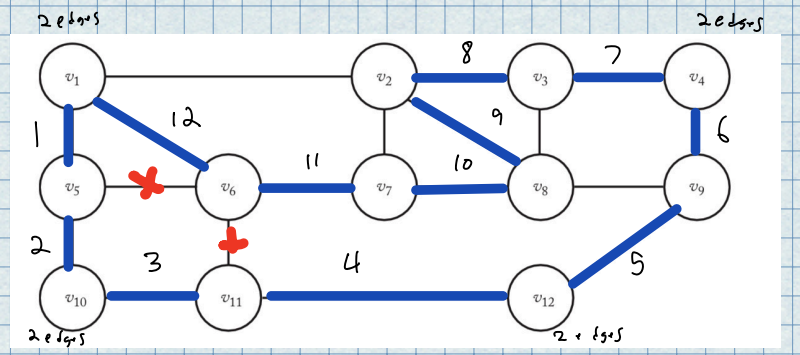


Question 3

V6 has 3 edges however it cannot go to V5 or V11 because they connect with V1 and V10 and V10 and V12 respectively which only have 2 edges going into them. Because of that, V1 must connect to V10 via V5 so V5 can’t connect to V6. V10 must connect to V12 via V11 so V11 can’t connect to V6. This means that V6 has only one realistic edge going into it breaking the possibility of a Hamiltonian circuit.

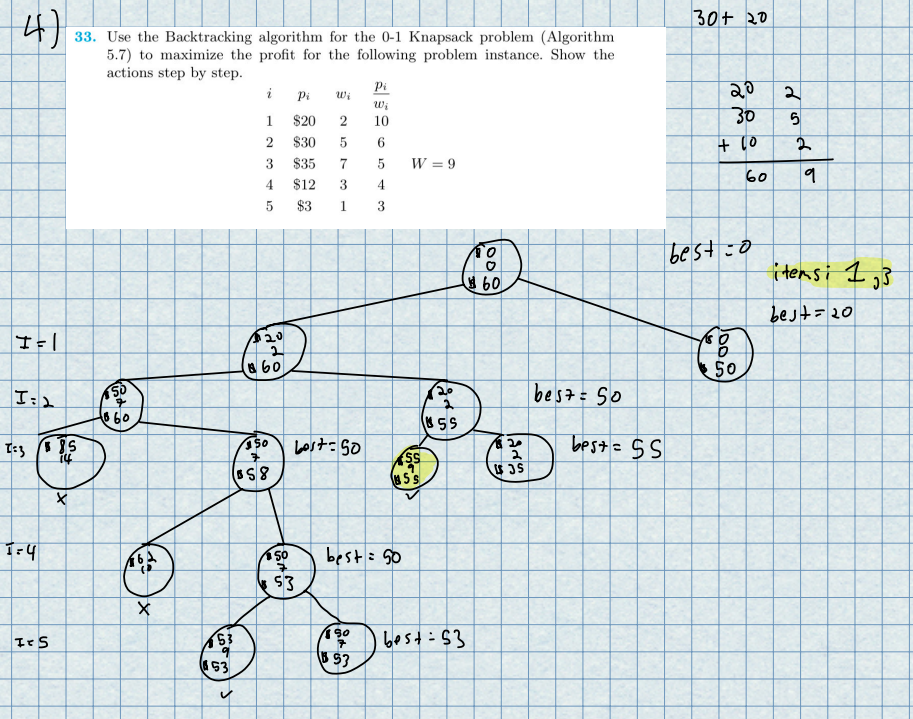


If an edge is created between v6 and v1 a Hamiltonian circuit can be formed.

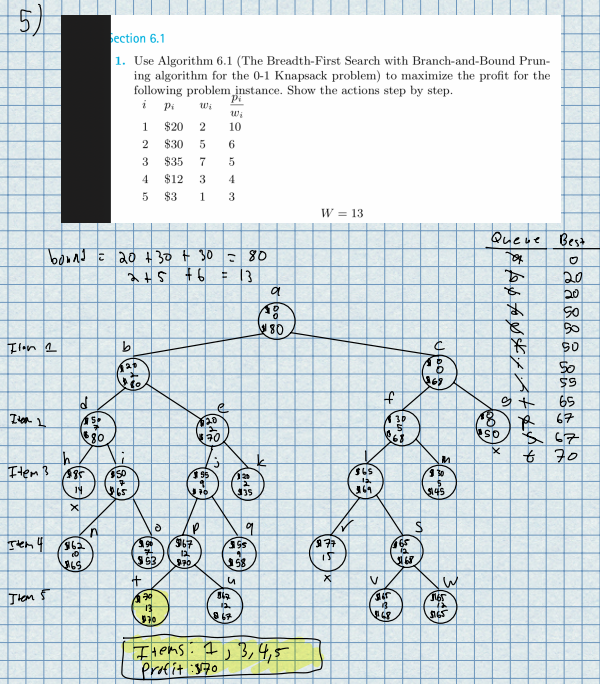


Question 4

Items: 1 and 3



Question 5



Question 6

