CMSC 3140:

Analysis Of Algorithms

Maximum Independent Set of Graph

Dec. 5th, 2024

Submitted By:

Margo B. Bonal

John Gerega

Luke Ruffing

Design:

Two algorithms were implemented to solve the maximum independent set problem. The two methods were a brute force approach and a greedy approach. Both algorithms are using an adjacency matrix as this method allows for easier searching of an edge. A matrix class was designed for this program as well as a user menu to prevent having two separate programs. The menu consists of 6 options: insert, print matrix, brute force, greedy, clear, and quit. The first two options are quite simple, the insert option will insert a 1 into random position in the matrix and the print option will print the current matrix. The user can only insert values one time, but they may print as many times as they please. The third and fourth options are a bit more complicated.

The first option that will be examined is the brute force approach, which is meant to search every row and column of the matrix searching for an edge. This process is started by the program calling the bruteMax() function. Within this function, we check if each vertex is independent using an isIndependent() function. This function goes through the matrix once more to determine that the row we are searching is independent by determining if there is a 1 in that position or a 0. If there is a 0, we return true from the function and if there is a 1, we return false. This brings us back into the bruteMax() function. If the isIndependent() function returns true, we add the vertex that was checked into an array called maxSet using a function named appendMax(). This function once again ensures that the vertex we have passed in is independent using the same function named before. If the vertex is independent, we add it to the array. Otherwise, we return back to the bruteMax function, and nothing is changed. This process is repeated until the end of the vertex is reached, at which point we go through the maxSet array to print out the vertices which are independent. This set is then displayed to the user as well as the time it took for the program to find this set. The brute force approach does help correctly identify

the maximum set but as the matrix gets bigger the time does as well, which is why the program also uses a greedy approach to solving the problem.

The greedy selection on the menu tries to find the maximum set in a shorter time than the brute force approach. The greedy() function starts when the user selects the number 4 on the previously mentioned menu. The function starts by clearing whatever numbers are in the max set and returning the values back to 0. The function then calculates the degrees of each vertex, or how many 1's are in each row of the vertex. The program then begins the greedy algorithm approach, setting variable minDeg to –1 and minDegV to V+1, with V being the size of the row. The program then goes through the matrix and checks if the current vertex is not included and checks if the degree at this position is less than the minimum. If both of these conditions are true, minDeg is set to the current degree and minDegV is set to the current index value we are on. If the loop completes processing and minDegV is still -1, the program breaks from the function. Otherwise, minDegV is appended to our maxSet. The program then sets included at the position of minDegV to be true. The program then enters another for loop that checks each vertex. If the position at minDegV, index j contains the value of 1, included at index j is set to true. After this loop processes, the program indexes through and prints out the maximum set. Once again, the user is shown the maximum set and the time it took the program to find this set.

The program will continue to accept options from the user until the number 6 is entered at which point the program is terminated. This allows the user to continue to manipulate the matrix by inserting values and seeing what the maximum independent set is based on their insertions.

Evaluation:

To compare the differences between the brute force approach and the greedy approach, multiple examples of different sized matrices were plugged into the program and the results are

recorded in the tables below. The first table shows the time for the brute force algorithm with different values of V.

- *Note: Does not include matrices that are extremely large*

Brute Force Algorithm			
Graph Size	**Time**	**Adjacency Matrix**	**Set**
V = 5	0.0007098 sec	0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0	Size = 3 Max Independent: {0, 2, 4}
V = 15	0.0008299 sec	0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0	Size: 7 Max Independent: { 2 4 6 8 10 12 14}
V = 35	9.62809 sec	-	Size: 20 Max Independent: { 0 2 3 4 10 11 12 13 14 16 19 20 23 24 25 27 28 29 31 34}
V = 40	184.309 sec	-	Size = 26 Max Independent: { 2 3 4 6 8 9 12 13 14 15 19 20 22 23 24 26 27 28 29 30 32 33 34 35 37 39}
V = 45	2222.92 sec	-	Size: 24 Set: {0 7 10 11 12 18 20 21 23 24 29 30 31 32 33 34 37 38 39 40 41 42 43 44}
V = 47	3117.67 sec	-	Size: 27 {0,1,6,8,9,10,12,13,20,21,23,24,25,26,27,29,31,32,36,37,38,39,40,41,42,44,45}
V = 65	Overnight	-	Inconclusive Result

As the table shows, the running time for brute force increases very quickly as the values of V continue to increase. While this approach will consistently provide the correct maximum independent set, it is best suited for small values of V, that way the running time does not become too large. Now, the greedy algorithm is shown below.

Greedy Algorithm			
Graph Size	**Time**	**Adjacency Matrix**	**Set**
V = 5	0.0001135 sec	-	{0,2,3}
V = 10	0.0004178 sec	-	{ 1 5 6 9 8 7}
V = 15	0.0005039 sec	-	{ 0 3 8 10 11 12 4 14 1}
V = 20	0.000901 sec	-	{ 2 6 7 9 10 16 18 19 0 4 1}
V = 25	0.0005494 sec	-	{ 0 2 5 6 11 12 17 18 20 22 23 10 14 19}
V = 30	0.0008573 sec	-	{ 0 2 3 4 5 6 7 13 15 16 17 22 23 25 28 12 20 27}
V = 40	0.0008107 sec	-	{ 0 1 6 11 15 17 19 20 23 26 27 28 31 32 35 3 5 12 22 29 10}
V = 45	0.0050606 sec	-	{0 2 11 12 15 18 20 23 30 31 33 34 36 42 1 6 22 24 26 29 32 34}
V = 47	0.0042119 sec	-	{1,6,8,9,12,13,20,24,25,26,31,32,38,39,40,42,45,14,18,21,23,29,41,44,0,35}
V = 65	0.0012632 sec	-	{ 1 3 5 8 9 10 16 20 27 29 31 32 38 41 45 46 47 48 53 57 58 59 64 0 4 15 21 22 23 25 35 36 55 62 2 60}
V = 85	0.0019994 sec	-	{ 0 3 10 13 23 24 26 30 32 35 40 42 47 48 51 54 58 59 61 66 69 70 71 75 76 81 84 2 4 5 7 15 16 19 22 27 31 37 43 46 56 57 9 60 82}
V= 100	0.0023002 sec	-	{ 1 4 7 11 13 20 24 29 30 32 33 34 36 44 45 49 53 58 59 60 62 65 69 72 73 75 81 82 84 87 90 91 94 95 96 98 0 9 14 15 17 25 27 39 40 46 48 50 54 56 61 74 79 92 35 85}
V = 500	0.0109183 sec	-	{ 0 3 6 8 9 11 18 19 21 23 26 28 30 31 32 35 37 38 46 48 49 50 51 53 54 55 60 61 64 71 72 73 75 79 85 86 90 92 94 107 108 109 111 116 117 119 120 121 122 123 126 131 132 134 135 136 137 138 141 150 151 155 158 159 168 169 184 185 187 188 192 194 196 197 203 210 211 214 215 216 218 219 222 223 225 226 229 235 239 240 242 243 244 250 251 258 259 261 262 264 267 268 279 282 285 291 292 294 296 300 302 304 305 307 309 313 314 315 316 319 320 322 324 328 331 336 338 340 344 346 348 350 351 352 353 354 357 360 364 366 368 376 377 380 381 382 386 388 390 391 392 395 398 404 406 407 411 412 413 415 416 421 425 428 431 432 434 443 444 446 447 448 451 460 469 470 478 480 484 485 494 495 496 4 7 12 14 20 22 24 25 40 47 56 59 63 67 69 70 74 78 80 87 88 91 98 101 130 146 156 157 160 162 164 167 171 177 179 198 200 201 209 213 221 233 234 236 238 241 246 253 254 256 265 266 270 271 272 276 280 318 327 332 335 358 361 371 372 375 385 389 393 396 401 408 418 423 438 439 440 464 472 477 479 492 499 58 89 144 183 231 277 325 337 445 95 170 149}

As shown in the greedy table, the greedy algorithm runs much faster than the brute force algorithm, however the sets found do not always match the ones found by the brute force

algorithm. This approach, while drastically cutting down on running time, does not guarantee that the set we receive is the maximum set. Contrary to the brute force algorithm, much larger values for V are allowed as the greedy algorithm is not searching the matrix as much as brute force, instead taking the most optimal decisions when determining the maximum set.

In Summary, both the brute force and greedy algorithms serve a specific purpose. They allow us to analyze small and large data sets relatively easy. They differ in performance and runtime. Greedy Algorithm is superior in runtime, handling an adjacency matrix with vertices of 47 in only 0.0042 seconds. Whereas the brute force algorithm takes approximately 51 minutes to process the same set. Through inspection, Greedy produces slightly differing max independent sets. This makes it not as reliable as the brute force algorithm that almost insures the proper max independent set. The brute force algorithm's runtime points to its method of checking every possible solution. For V vertices, there are 2^V subsets. This makes its runtime O($2^V * V^2$). This is an exponential runtime, proving large data for V infeasible. As seen above in our Brute Force table, runtime jumps drastically from V = 35 to an inclusive result at V = 65. This computationally expensive runtime makes the brute force practice for only vertices under 35. Likewise, the Greedy Algorithm's runtime is significantly different. It runs in worst case, polynomial time of $O = (V^2)$ since finding the vertex with smallest degree among unprocessed vertices takes O(V) two times. Comparing these two runtimes shows that Greedy is more efficient. By implementing a graph built by adjacency matrix, a brute force and greedy algorithm, we can see the limitations of our modern computing power and search for ways of revision.

Resources:

Geeks For Geeks. (2024, Nov 5). ***Greedy Algorithms.***

https://www.geeksforgeeks.org/greedy-algorithms/

Geeks For Geeks. (2024, May 28). ***Implement Adjacency Matrix.***

C++ Program to Implement Adjacency Matrix - GeeksforGeeks

Microsoft. (2022, April 5). ***<Chrono>.***

https://learn.microsoft.com/en-us/cpp/standard-library/chrono?view=msvc-170