Kevin Martin – HW5

Syracuse University CIS 675 – Thursday @ 9:00pm EST

3/12/2020

**Question 1**

a) The worst case running time of a sequence of *n* operations by using the aggregate method of amortized analysis is as follows:
= $1*n/3 + 2*n/6 + 3*n/9 + \ldots + (3^{(\log n)}/3)*n/3^{(\log n)} + n * 3$ [when it's a power of 3]
= $n/3 + n/3 + n/3 + \ldots + n$
= $(n/3) *(\log n) + n = (1/3)(n \log n) + n$
= **O(n log n)**

b) The amortized cost per operation is O(n log n)/n = **O(log n)**

**Question 2**

a) The worst case running time of a sequence of n nextSubset() operations via amortized analysis using the aggregate method of amortized analysis is essentially the number of times *x[i]* changes from 0 to 1 (or vice versa). The number of times *x[n]* changes is equal to *m*, therefore x[n-1] changes at least m/2 times, x[n-2] changes at least m/4 times, and x[n-3] changes at least m/8 times. Thus each change for x[n-i] changes at least $m/2^i$ times. The total changes for *m* will be at least $(\log 2\ m)^{m/2i}$, which, when simplified will be less than 2m. Ignoring the constant, this gives an amortized running time of *O(m)*.

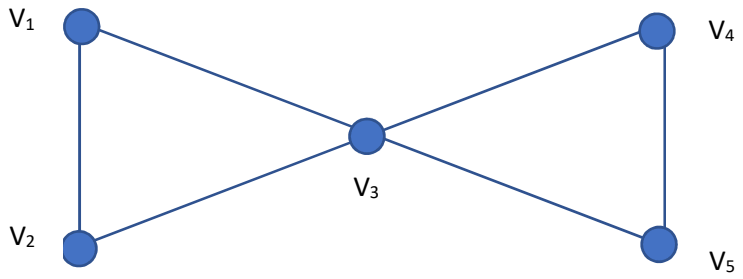
b) The amortized cost per nextSubset() is *O(m)/m,* which gives us constant time *O(1)*.

**Question 3**

We cannot prove that a given graph G has a double-Hamiltonian tour is NP-Complete, because we cannot prove that there is an NP solution to it. We can show that the decision is NP-Hard by a reduction from the regular (single) Hamiltonian cycle. When we construct the new graph *H* by attaching the new vertices vx and vy, and the edges (v,vx), (vx, vy), and (vy, v). **If** G has a Hamiltonian cycle $v_1$->$v_2$->...->$v_n$->v1, then we can build a double-Hamiltonian tour of graph H by replacing each vertex $v_i$ with the following: $v_i$->$v_i^x$->$v_i^y$-> $v_i^x$->$v_i^y$->$v_i$->… .

To show this going the other way, if *H* has a double-Hamiltonian tour T, then any vertex v in the original graph G will be visited exactly twice. This means that T is two separate closed walks, each of which visits v only once. Any walk from $v^x$ or $v^y$ to any other vertex in H must pass through v. Thus, one of T's walks visits only v, vx, and vy. If we remove the new vertices $v^x$ and $v^y$ to construct H from the tour T, then we obtained a closed walk in G that visits every vertex v at least once. G has a Hamiltonian cycle if and only

if H ahs a double-Hamiltonian circuit. Furthermore, if we added a graph P that was itself a double-Hamiltonian circuit to a vertex in graph G, it will traverse each edge of H if and only if G contains a single Hamiltonian cycle. This proves the problem is NP-Hard. However, we cannot show the problem is NP-Complete. If we instead construct the new graph H by simply attaching a self-loop at every vertex of the original graph G, we can alternate between edges of the Hamiltonian cycle of G and the self-loops, creating a double-Hamiltonian cycle. Importantly, we cannot guarantee that double-Hamiltonian circuit in H does or does not use any self-loops. Consider the below graph as a counterexample: it has a double-Hamiltonian circuit, but no Hamiltonian cycle.




**Question 4**

a) To design an algorithm that gives satisfying assignment to the 2SAT problem where each clause is the disjunction of two literals, we can consider a graph of all the clauses. For every clause ($X_1$ V $X_2$), two edges are created such that one is ($\overline{X}_1$ -> $X_2$) and the second is ($\overline{X}_2$ -> $X_1$). If any two variables are present in the same cycle, then the 2SAT problem will be false. If, however, the two variables are not in the same cycle, than at least that clause will be true. The algorithm simply needs to check if the two edges are present, and if so return FALSE, otherwise mark as TRUE and keep checking:

```
def satCheck(G)
nodes[n]                                    // n equals all the nodes in the graph
for (i = 1 to n) do:                        // mark all nodes as unvisited
   nodes[i] = -1                            // -1 indicates not visited
for (j = 1 to n) do:
   if (visited(nodes[j]) == -1) then:       // visit all notes not already visited
      nodes[j] = 1                          // 1 indicates visited
      for (int k = 1 to length(neighbors(j)) do:   // visit all neighbors of j
         visit(k)
      G.add(j)                              // add j to graph
for (p = 1 to length(G)) do:                // set up the graph with edge values
   if (visited(nodes[p]) == -1) then:
      for (int q = 1 to length(neighbors(j)) do:
         visit(q)
for (t = 1 to length(G)) do:                // 2SAT check
   if (G[t] == G[t+1]) then:                // indicates the presence of x and x
      return FALSE                          // the 2SAT is unsatisfiable, exit
return TRUE                                 // the SAT is satisfiable
```

b) The running time of this algorithm is *O(V + E)*, where V is the number of vertices and E is the number of edges. This is because each edge and each vertex must be checked using a depth-first search approach. The space complexity is *O(V)* as only the results of the vertex checks need to be stored.

**Question 5**

a) The k-degree spanning tree is a search problem because we are not looking for an optimal solution, we are simply trying to verify a possible solution with a given requirement (in this case, that each vertex has a degree of less than or equal to *k*). This can be done in polynomial time using a dept-first search and checking that the degrees are not greater than *k*.

b) To show that the k-degree constrained problem is NP-complete for any $k \geq 2$, we can start with the fact that for a spanning tree with a degree equal to 2, that is a Hamiltonian path. So, we can reduce the problem from the Hamiltonian path problem in polynomial time by arranging the vertices to be a spanning tree with degree equal to 2. We can also revert back to our original problem as well in polynomial time, by simply undoing the movements. If graph G has a Hamiltonian path, we can connect each vertex to $k - 2$ new vertices, creating graph H. Graph H is now a k-degree spanning tree. Graph H will now solve our original problem. Similarly, by solving H for a degree equal to 2, we can also solve a Hamiltonian path problem, so the process works both ways. By reducing the Hamiltonian path to the spanning tree problem, we have shown that the problem is NP-complete.