HW 5 CSE 541

Qn.1

If we have a graph G as a ham-graph, which means that there is a ham-circle in this graph G.

Then let’s choose any vertex v from G (the graph), and consider all the possibilities of deleting all but two of the edges passing through that vertex.

One as go into this vertex and one go out of this vertex.

And the resulting graph must still be ham-graph since the ham-circle that existed originally only used two edges (for three vertices). Since the degree of a vertex is bounded by

(the number of vertices -1), so we only less than (number)2 by go through all pairs which is O(n2).

So, we can see that we are only running the polynomial tester polynomial many independent times, so the running time is polynomial.

Once we have some pair of vertices where deleting all the others coming off v still results in a ham-graph, we should remember those as special, and ones that we will never again try to delete. We repeat the process with both vertices that are now next to v, testing ham of each way of picking a new vertex to save, we can continue in this process until we are left with only |V| edge, and then, we just show listing the vertices of a ham-cycle is polynomial-time.

Qn.2-1 (34-5-2)

**show that 3-CNF-SAT ≤P 01LP**.

Proof:

Let φ be 3-CNF which has n input variables and m clauses.

We construct an instance of 01LP as follows.

Let A be a m + 2n by 2n matrix.

For 1 ≤ i ≤ m,

if (1 ≤ j ≤ n && clause Ci contains the literal xj : set entry A(i, j) to -1

Otherwise set it to 0.

For n+1 ≤ j ≤ 2n,

if clause Ci contains the literal ¬xj−n : set entry A(i, j) to -1

Otherwise set it to 0.

When m + 1 ≤ i ≤ m +n,

if i−m = j or i−m = j –n: set A(i, j) = 1

else: 0

When m+ n + 1 ≤ i ≤ m + 2n,

if i − m− n = j or i − m − n = j − n: set A(i, j) = −1

else: 0

for example:

A :[-1 -1 -1 0]

…..[-1 0 1 1]

Let b be a m + 2n-vector.

Set the first k entries to -1, the next n entries to 1, and the last n entries to -1.

b [-1]

… [1]

**So, we can conclude that the time of constructing A and b is polynomial time.**

Reduction Algorithm

Algorithm F(φ)

Check whether φ is in 3-CNF format if it is the return (A =[1], b=[2]), let m be the number of clauses and n be the number of variables in φ

For i =1 to m:

NUM <- 0;

For j=1 to m: set A[i,j] =0

For each literal L in Ci:

If L =xj, then

A[i,j] =a [i,j] +1;

NUM++;

Else if 1 =7xj then:

A[i,j] = a[i,j] -1;

NUM++

End if

End for

b[i] =1 –NUM;

End for

Return (A,b)

Statement:

If φ is not in 3-CNF format then A=[1] and A =[2], and then system 1x1>=2 has no solution for x1 =1 or 0. It remains to look at the case that φ is in 3-CNF format. In that case, φ is satisfiable if and only if there exists a truth assignment to x1,x2..xn such that this is true if and only if there exists vale 0, 1 assgined to variable x1, x2… xn such that Ax>b, let us analyze each clause Ci. Since clause Ci is satisfied by x1,x2,xn iff at least one of those is assigned to be true

Qn.2-2 (34-5-3)

Since the 0-1 integer-programming problem is NP-hard, we need to show a reduction from the integer to 0-1 problem.

Proof:

If we take the A from 0-1 LP problem, and tack on a copy of the n x n identity matrix to its bottom, and track on n ones to the end of b from the 0-1 integer-programming problem. This has the effect of adding the restrictions that every entry of x must be at most 1. But, since for every i, we need xi to be an integer rather than just 0 or 1, this only leaves the option that xi =0 or xi=1. This means that by adding these restrictions, we have that any solution to this system will be a solution to the 0-1 integer programming problem given by A and b. let y1y2y3 correspond to the truth values of each literals in Ci. So, Ci is satisfied iff at least one of y1y2y3 = 1, which is equaling to saying y1+y2+y3 >=1.

Now we relate be values of y1y2y3 with the values of x1x2 xn. Let l1 l2 and l3 be the literals in Ci. Then if li =xkj then yi= xkj, but if lj = 7xkj then yi = 1-xkj since y1 =1 if xkj =0 and y1 =1 if xkj =1. Substituting this into the equation y1+y2+y3 >=1 we get ak1\*xk1 + ak2\*xk2+ ak2\*xk2 >=1-ni

Where aki = 1 if li =xkj or =0 if lj =7xki

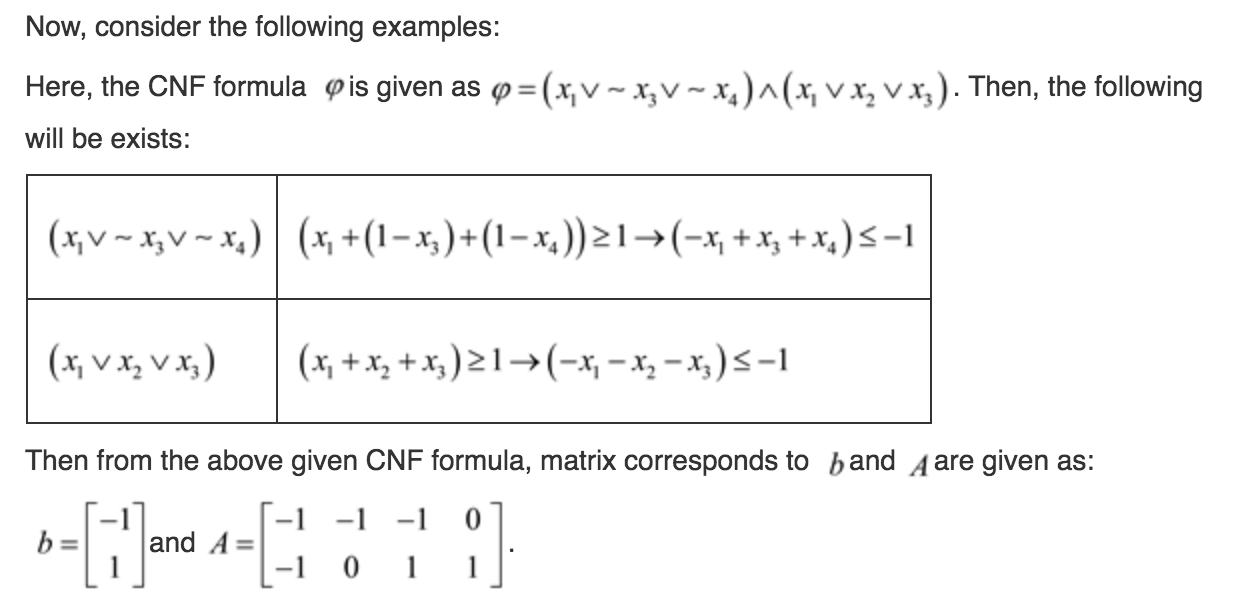
And ni is the number of variable in Ci that appear negated. That φ is satisfiable if and only if there exists a 0-1 assinment variables x1….xn such that Ax>=b.

Run in polunomial time:

Checking whether φ is in 3-CNF format can be done in linear time on the number of clauses. The loop on i runs in m steps, the loop om j runs in nsteps the loop on the literals run in constant number of steps, since there are only 3 literals per clause. So the running time of the function is O(nm), which is polynomial time.

Qn.2-2 (34-5-3)

The 0-1 integer programming problem will be proved as NP by using the face that 3-CNF-SAT <= 0-1 integer programming problem.

Let we set a CNF formula φ consists of n variables and m clauses. And build A and b as following:  


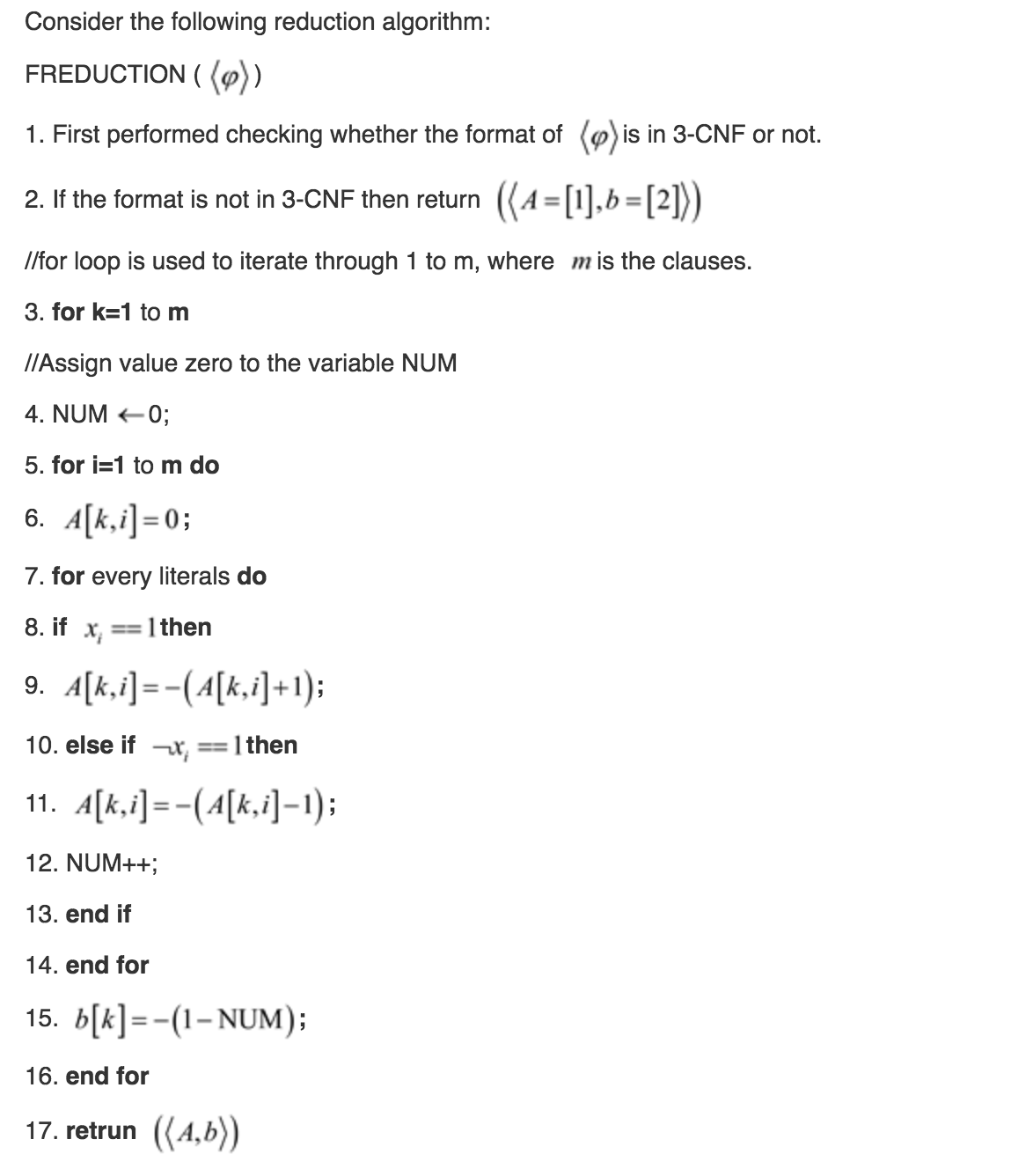
Use the algorithm in the next page to find the format of φ.

Now, consider if the format of φ is not in 3-CNF then (A=[1] b=[2]) and the system has no solution for xi ∈0,1.

In that case, if there exists a truth assignment to x1x2xn in such way that every clause is satisfiability then φ is said to be satisfiability.

Then it is true if there exist the values 0,1 assgined to variable x1x2xn in such way that Ax<=b.

Hence, form the above explanation it is proved that integer programming problem is NP-complete.



Qn.3 (34-5-8)

A certificate would be an assignment to input variables which causes exactly

half the clauses to evaluate to 1, and the other half to evaluate to 0.

Since we can check this in polynomial time, half 3-CNF is in NP.

**To prove that it’s NP-hard, we have to show that 3-CNF-SAT ≤p HALF-3-CNF.**

Proof:

Let φ be any 3-CNF formula with m clauses and input variables x1, x2, .. , xn. Let T be the formula (y∨y∨¬y), and let F be the formula (y∨y∨y).

Let φ’ = φ∧T ∧. . .∧T ∧F ∧. . .∧F where there are m copies of T and 2m copies of F. Then φ’ has 4m clauses and can be constructed from φ in polynomial time.

Suppose that φ has a satisfying assignment. Then by setting y = 0 and the xi

’s to the satisfying assignment, we satisfy the m clauses of φ and the m T clauses, but none of the F clauses. Thus, φ’ has an assignment which satisfies exactly half of its clauses. On the other hand, suppose there is no satisfying assignment to φ. The m T clauses are always satisfied. If we set y = 0 then the total number of clauses satisfies in φ’ is strictly less than 2m, since each of the 2m F clauses is false, and at least one of the φ clauses is false. If we set y = 1, then strictly more than half the clauses of φ’ are satisfied, since the 3m T and F clauses are all satisfied.

Thus, φ has a satisfying assignment if and only if φ’ has an assignment which

satisfies exactly half of its clauses. We conclude that HALF-3-CNF is NP-hard,

and hence NP-complete.

Qn.34(34-1)

a)

Decision problem:

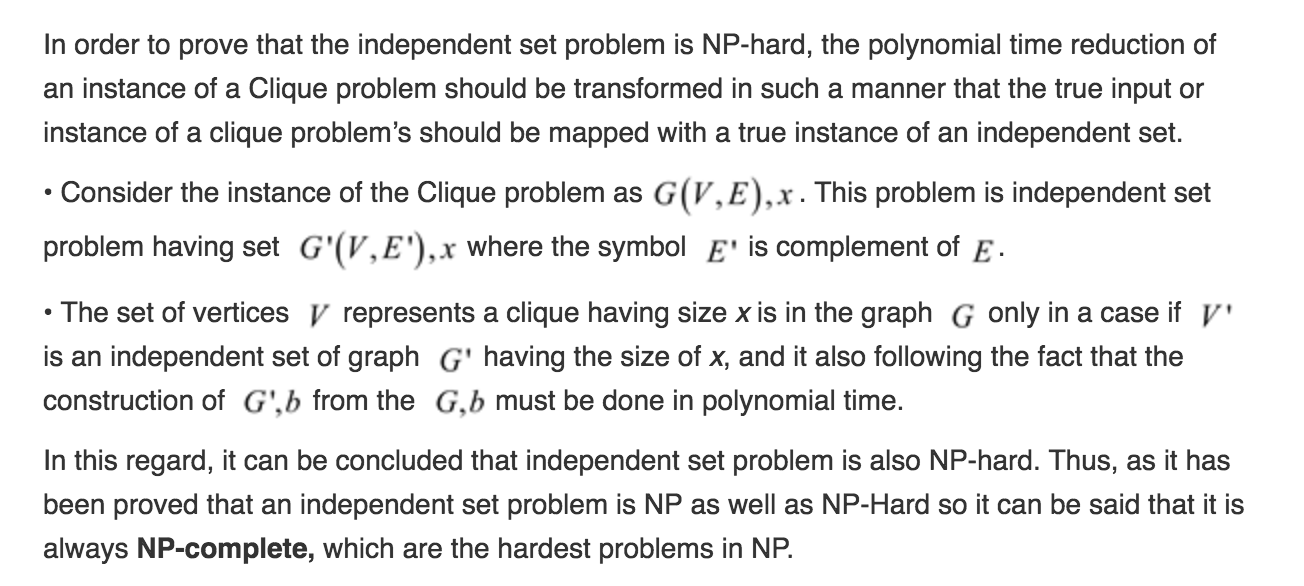
Is there some independent set of (sizes at least K) given a graph and a number k?(1)

If we take the compliment of the given graph, then it will have a clique of size at

least k if and only if the original graph has an independent set of sizes at least k.

*which means subset <= original set*

Since if we take any set of vertices in the original graph, then it will be an independent set if and only if there are no edges between those vertices.



However, in the compliment graph, this means that between every

one of those vertices, there is an edge, which means they form a clique.

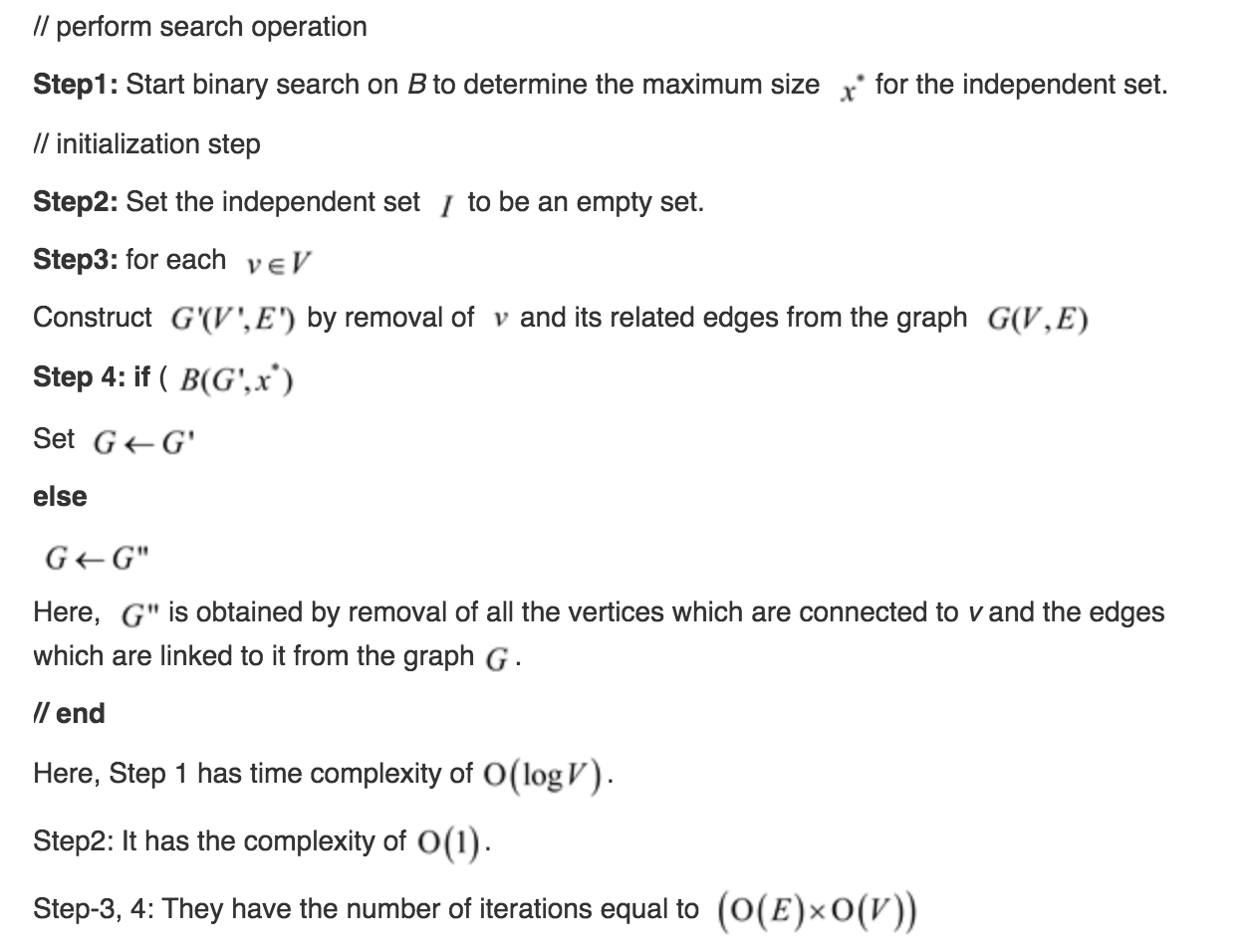
So, to decide independent set, just decide clique in the compliment.

Then we can say that decision problem (1) is NP-complete.

b)

given a black-box subroutine to solve the decision problem define in part(a) and a graph G(V,E). here, it is asked to give an algorithm which can be used to find a independent set of max size. We set the black-box a B(G,x)

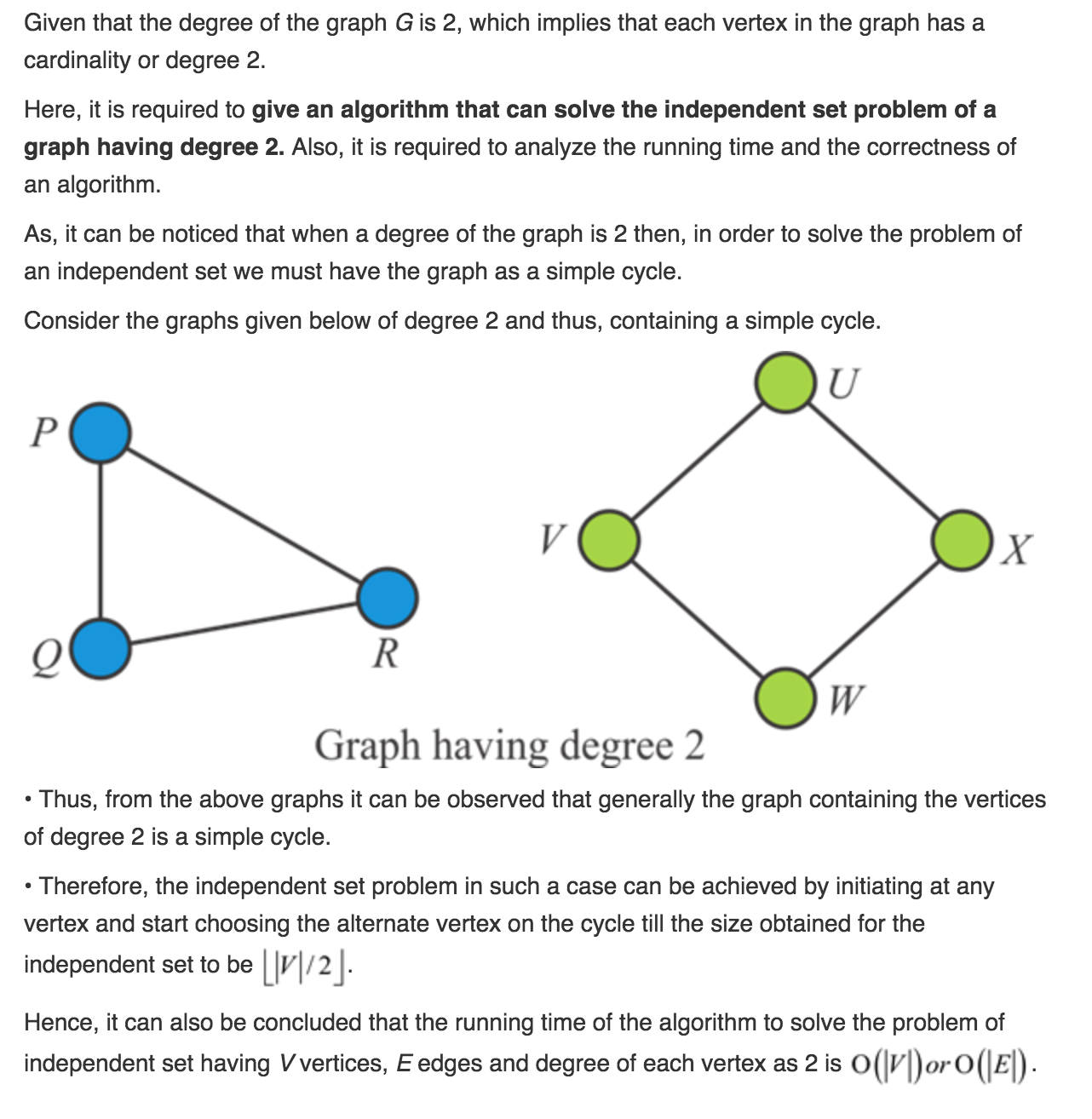
Algorithm:

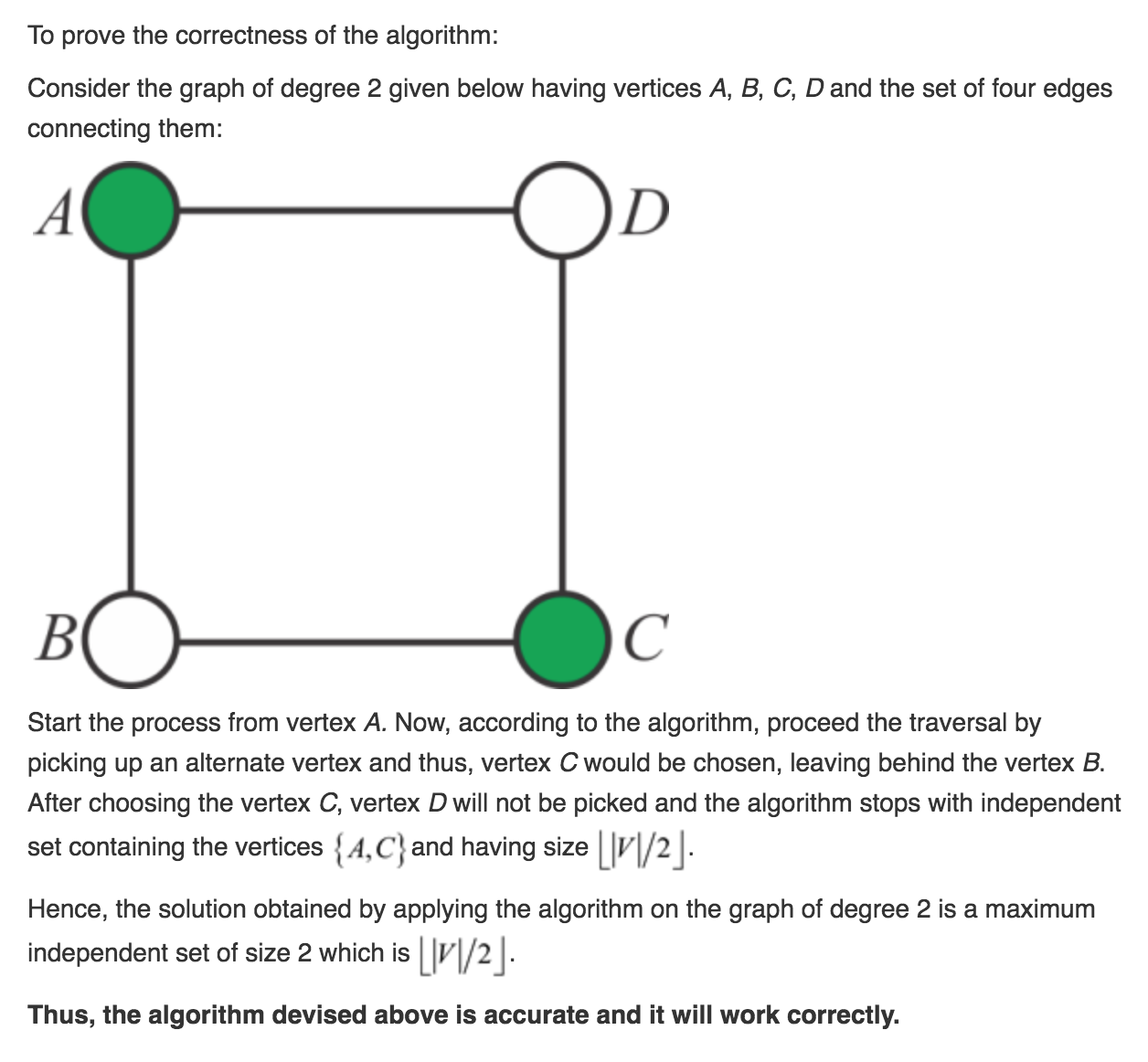


hence, the vertex in G which is obtained at last is independent set of sizes X\* by the construction.

Hence, it can be concluded that the time complexity is O(V+E).

c)





d)

First, find a maximal matching. This can be done in time O(V E) by

the methods of section 26.3. let f(x) be defined for all vertices that were

matched, and let it evaluate to the point that is paired with x in the maximal

matching. Then, we do the following procedure. Let S1 be the unpaired

points, Let S2 = f(N(S1)) \S1, where we extend f to sets of vertices by just

letting it be the set containing all the pairs of the given points. Similarly,

define Si+1 = f(N(Si)) \(Ui j=1 Sj).

First, we need to show that this is well defined. That means that we want to make sure that we always have that every neighbor of Si is paired with something. Since we could get from an unpaired point to something in Si by taking a path that is alternating from being an edge in the matching and an edge not in the matching, starting with one that was not, if we could get to an unpaired point from Si

, that last edge could be tacked onto this path, and it would become an augmenting path, contradicting maximalist of the original matching. Next, we can note that we never have an element in some Si adjacent to an element in some Sj .

Suppose there were, then we could take the path from an unpaired vertex

to a vertex in Si, add the edge to the element in Sj and then take the path

from there to an unpaired vertex. This form an augmenting path, which

would again contradict maximalist. The process of computing the {Si} must

eventually terminate by becoming Φ because they are selected to be **disjoint**

and there are only finitely many vertices. Any vertices that are neither in an

Si or adjacent to one consist entirely of a perfect matching, that has no edges

going to picked vertices. This means that the best we can do is to just pick

everything from one side of the remaining vertices. This whole procedure of

picking vertices takes time at most O(E), since we consider going along each

edge only twice. This brings the total runtime to O(V E).