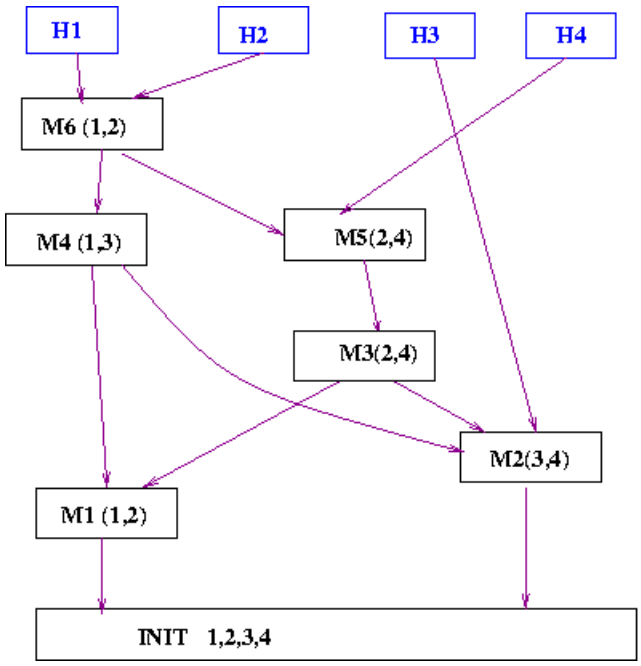
Tutorial 8 Solutions

1. The graph is an extremely useful modelling tool. Here is how a COVID tracing tool might work. Let V be the set of all persons. We say (p,q) is an edge (i) in E1 if their names appear on the same webpage, and (ii) in E2 if they have been together in a common location for more than 20 minutes. What significance do the connected components have in these graphs, and what does the BFS do? Does the second graph have epidemiological significance? If so, what? If not, how would you improve the graph structure to get a sharper epidemiological meaning?

   Solution:

   The first graph, with edges E1, gives us a measure of the closeness of two people with each other. This may be interpreted as a measure of how frequently they come in contact with each other, and clearly the more often they come in contact, the more likely the virus is to spread from one of them to the other. Thus, we can use this as a kind of contact tracing tool; BFS gives us the shortest path between any two people, and the shorter this path is, the more likely it is that the virus has spread from one end to the other. For every infected person, we can do a BFS over the graph, and all nodes (people) having small shortest distances from that infected person can be classified as having high risk. Any connected component thus consists of a number of people who have high chances of spreading the virus among themselves if any one gets infected. This can be used to determine the risk faced by every individual in the system, due to a particular person being infected.



The second graph, with edges E2, can similarly be used to identify high risk individuals, since any infected person is likely to have spread the infection to their neighbour(s) in the E2 graph, in the 20-minute interval where they were close together. However, this information is not sufficient for points separated by more than one edge, because the graph does not contain any information about the chronology of the 20-minute interactions. For example, for two individuals separated by three edges, the virus would only spread from one end to the other if the 20-minute interactions occurred in the right order. So, we can improve this data structure, by additionally maintaining a timestamp on each edge, indicating the time at which the 20-minute interaction took place.

The vertices are V={ [Mi,Si]| i-th meeting label, Si=people who participated}. There is an edge from [Mi,Si] to [Mj,Sj] iff (i) there is a person p common to Si and Sj, (ii) i<j, and (iii) p did not participate in any meeting between i and j.

How is one to generate such a graph:
Let header(p) point to the last meeting in which p participated. Suppose a new meeting Mk,Sk happens with Sk={p,q,r} then the code is:

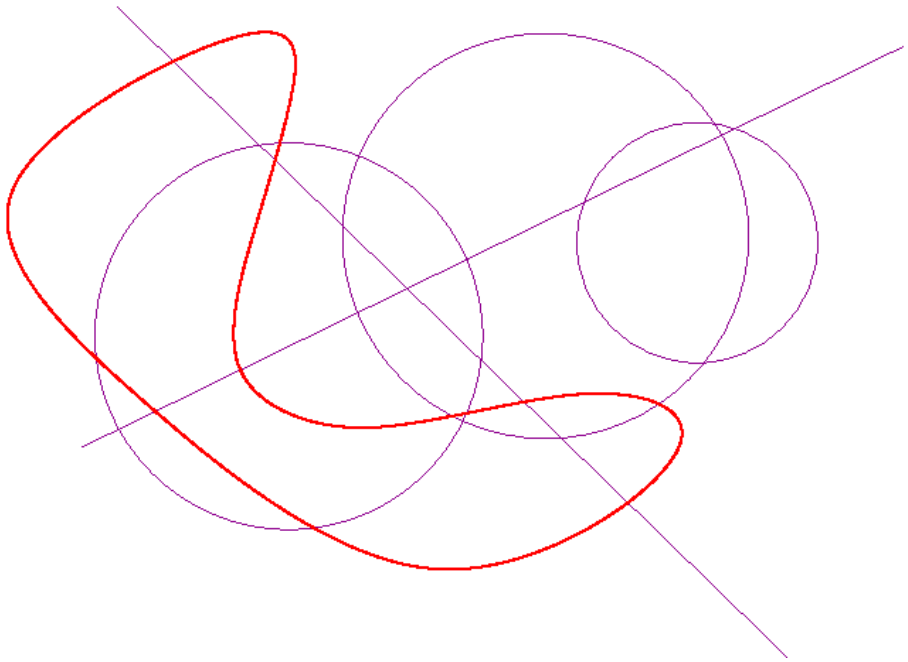
```
Function insert(k,Sk)
New meeting m;
m.S=sk;
m.id=k;
m.next=[ ];// adjacency list
For all p in Sk
  m.next=append(m.next, header(p);
  header(p)=m;
Endfor
Endfunction
```

Understand how this function works. Suppose you know that between meeting 34 and 35, person p go covid. You are the person q. Use BFS to detect if you are at risk.

2. Let us take a plane paper and draw circles and lines to divide the plane into various pieces. There is an edge (p,q) between two pieces if they share a common boundary of intersection (which is more than a point). Is this graph bipartite? Under what conditions is it bipartite?

   Solution:

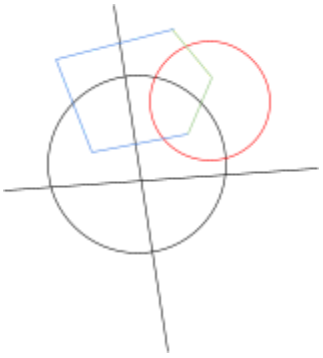
Yes the graph is bipartite. We can prove this by induction on the number of objects (say n).
Base Case (n = 1) : Either a line or a circle exists which divides the plane in 2 regions, adjacent to each other ($K_{1,1}$ is obtained).



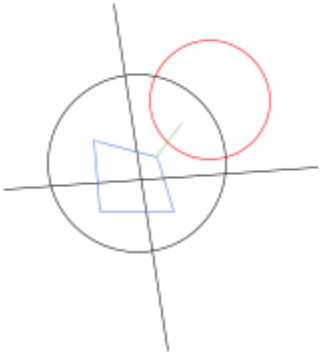
Inductive Hypothesis : The graph obtained using n objects is bipartite.
Proof : Let there be an arbitrary graph G obtained using any n + 1 objects. Pick any object and remove it. Let the resulting graph be G'. We know by inductive hypothesis that G' is bipartite i.e. it doesn't contain any odd length cycle. Now add the $(n+1)^{th}$ object (say p) to get G. This object might be a line or a circle. Nevertheless it will divide the plane into 2 parts (2 parts on either side of a line or 2 parts inside and outside a circle). We need to prove that the G doesn't contain any odd length cycle. Consider an odd-length cycle in the new graph. We can construct a geometric version of it as shown. Now remove the last object. By induction, it must then be an even cycle. Now consider any even length cycle which intersects the new object p. It will intersect even regions present on the cycle.

Case 1 : It intersects 2 regions of 2*L length cycle.

Each of the regions will be divided into 2 smaller regions, resulting in a new cycle of length 2L + 2. So the new cycle will always be even.

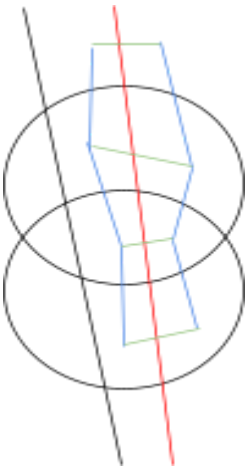


Case 2 : It intersects 1 region of 2*L length cycle.
The new cycle will remain as is because one of the 2 new regions obtained will never participate in the cycle.



Case 3 : It doesn't intersect any regions of a cycle.
This can be safely ignored.

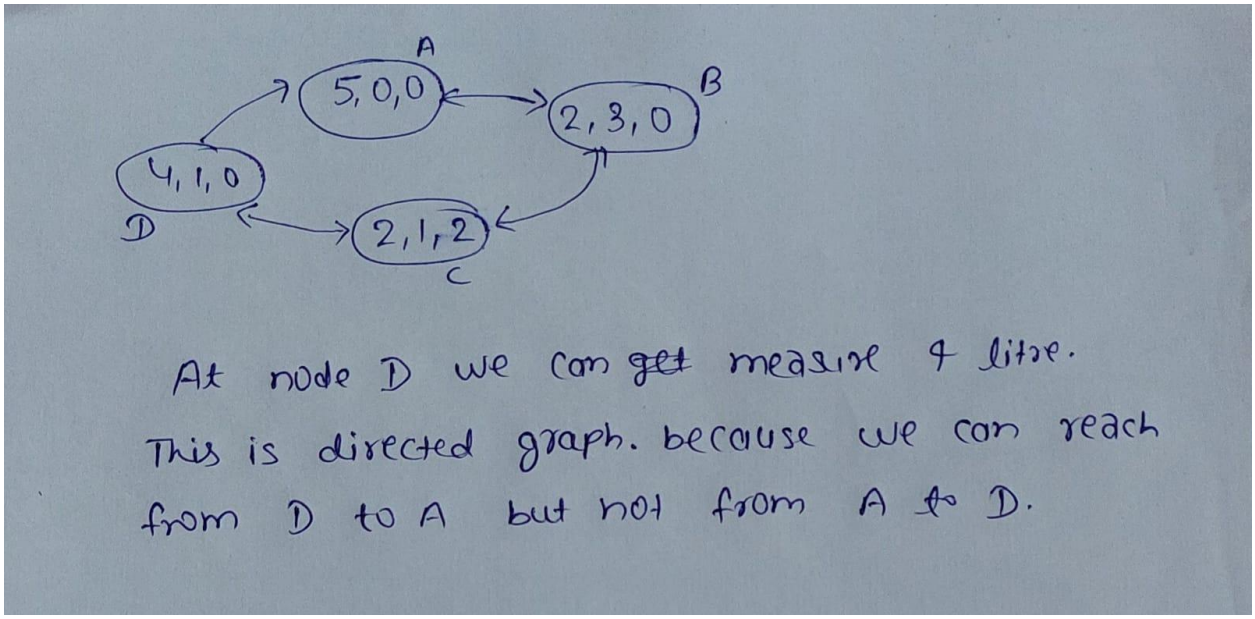
We also need to prove that newly formed cycles are always even length. New cycles are obtained only when p divides a set of regions forming a path, making exactly 2 copies of such a path, each on either side of p. Let the length of any such path be L. All the resulting cycles obtained will always be of length 2 + 2*k, which is even (where 1 <= k <= L).



Therefore, no new cycles of odd length are present in G. Hence G is bipartite.

3. There are three containers A, B and C, with capacities 5,3 and 2 liters respectively. We begin with A having 5 liters of milk and B and C being empty. There are no other measuring instruments. A buyer wants 4 liters of milk. Can you dispense this? Model this as a graph problem with the vertex set V as the set of configurations c=(c1,c2,c3) and an edge from c to d if d is reachable from c. Begin with (5,0,0). Is this graph directed or undirected? Is it adequate to model the question: How to dispense 4 liters?

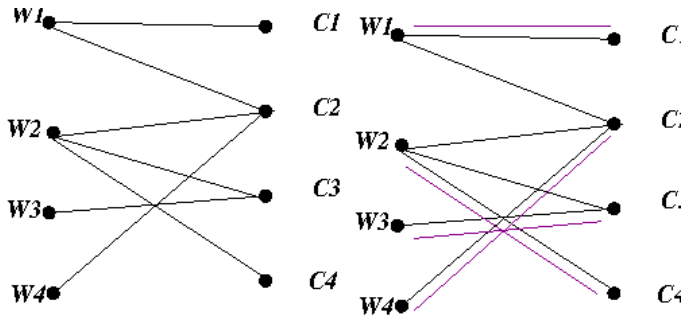
   Solution:



4. Suppose that there are M workers in a call center for a travel service which gives travel directions within a city. It provides services for N cities - C1,...,CN. Not all workers are familiar with all cities. The number of requests from each city per hour are R1,...,RN. A worker can handle K calls per hour. How would you model this problem? Assume that R1,...,RN and K are small numbers.

   Solution:

   This problem can be modelled as finding the optimal matching between the set of workers and the set of cities such that most of the requests are handled.
   Here both the worker and city can be denoted as two sets of nodes that are bipartite. And a matching is defined as connecting a worker vertex to a city vertex with which the worker is familiar. Also, we can have multiple workers being mapped to the same city or none of the workers being mapped to a particular city.
   We can solve this problem by first generating a bipartite graph having edges between the worker vertex and city vertex such that the worker knows the city.



Now the first step is to map workers having degree one to their corresponding cities as they have no other alternative.
After this decrease the request of that city by K. Now if the request of a city has become 0 then remove that city. Also remove the workers that have been mapped to the cities. Now recursively do this until no more cities can be removed.
If there are no single degree vertices present now then traverse all the adjacent cities of a worker and assign the city with the largest number of requests to it and do the same for other workers.
As we can see it contains overlapping subproblems so we can solve it by using dynamic programming.

5. There is a set of bureaucrats B={b1,...bm}. Subgroups of them keep meeting and making decisions of n attributes, e.g., parking is to be allowed (Y/N), Garba can take place (Y/N), etc. Let us call these as boolean variables P1,P2,...,Pn. Each meeting M has a time-stamp and a decision to change these boolean values. The new assignment is carried forward with the bureaucrats who participated. Model this as a graph. What questions can be answered using this model? For example, given a meeting in which two bureaucrats bi and bj have opposite decisions on an attribute, can they decide who of the two has the latest information?

   Solution:

   This is addressed in Problem 1.

6. Study the BFS code from Prof. Naveen's slides. Argue that at any time during the running of the algorithm the d-values of vertices in the queue can only take 1 or 2 consecutive values.

   Solution:

BFS algorithm starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level. Let's assume at time=k q1,...,qn are present in the queue and atmost 2 consecutive d-values are present.

Case-1:
Only one d-value (let's say x) is present. Then at time=k+1 q1 will be removed and it's unvisited neighbors would be pushed into the queue. The d-values of all the pushed neighbors would be x+1, thus maintaining our original assumption that the queue contains atmost 2 consecutive values at time t=k+1.

Case-2:
2 consecutive d-values are present in the queue. Then q1 to some qk would have d-value x and qk to qn would have d-value x+1. In the next step when we remove q1 and push all it's unvisited neighbors into the queue, the neighbors would have d-value of x+1 and our assumption would still hold.

At time t=0, we'll push the source vertex into the queue whose d-value would be 0. Hence, using induction we can say that the queue will have atmost 2 consecutive d-values at any point of time.

7. List the properties of the white, grey and black vertices. In line 10, while processing u, can we encounter a vertex v which is gray? Or black? In such cases what are the values possible for d[v]?

```
BFS(G,s)
01 for each vertex u ∈ V[G]-{s}
02    color[u] ← white
03    d[u] ← ∞
04    π[u] ← NIL
05 color[s] ← gray
06 d[s] ← 0
07 π[u] ← NIL
08 Q ← {s}
09 while Q ≠ ∅ do
10    u ← head[Q]
11    for each v ∈ Adj[u] do
12       if color[v] = white then
13          color[v] ← gray
14          d[v] ← d[u] + 1
15          π[v] ← u
16          Enqueue(Q,v)
```

```
17    Dequeue(Q)
18    color[u] ← black
```

Solution:

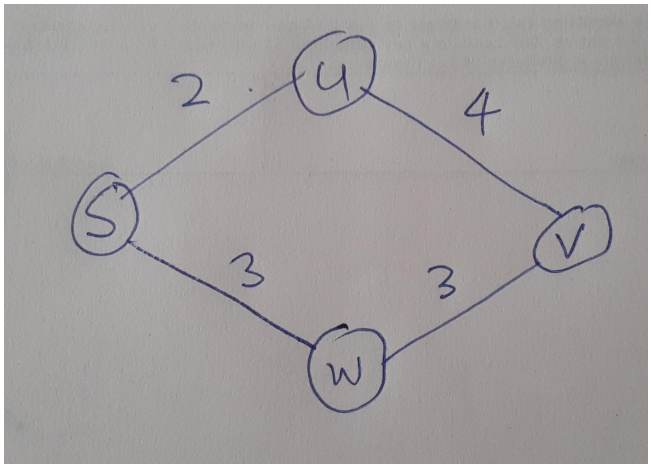
White vertices : Unvisited Vertices
Gray Vertices : Vertices under consideration (In Queue)
Black Vertices : Processed Vertices

We can encounter all types of vertices.

White vertice : If any child of the current node under consideration has an unvisited child.
D[v] = D[u] + 1
Black vertice : In adjacency list we can encounter parent of current node which is already processed.
D[v] = D[u] - 1
Gray vertice : Siblings at same level.
D[v] = D[u]

If one were to pause the execution of the algorithm, partition the set of vertices as V(white), V(grey) and V(black). Let S(white), S(grey) and S(white) be the set of distances of the vertices which are white, grey and black. Then there is a d such that all white vertices have distance from s greater than or equal to d+1. All black vertices have distance less than or equal to d.

8. There are many variations of BFS to solve various needs. For example, suppose that every edge e=(u,v) also has a weight w(e) (say the width of the road from u to v). For a path p= (v1,v2,...,vk), let the weight w(pi) be the minimum of the weights of the edges in the path. We would like to find all shortest paths from a vertex s to all vertices v. If there are multiple such paths, we would like to find whose weight is maximum. For example, in the graph below, we would prefer path s->w->v. Can we adapt BFS to detect this path?



Solution:

```
BFS(G,s)
01 for each vertex u ∈ V[G]-{s}
02    color[u] ← white
03    d[u] ← ∞
04    π[u] ← NIL
04B   width[u]=0; // path of width 0 exists to every node
05 color[s] ← gray
06 d[s] ← 0; width[u]=infinity
07
08 Q ← {s}
09 while Q ≠ ∅ do
10    u ← head[Q]
11    for each v ∈ Adj[u] do
12       if color[v] = white  then
13          color[v] ← gray
14          d[v] ← d[u] + 1; width[v]=min(width[u],w[(u,v)]);
15          π[v] ← u
16          Enqueue(Q,v)
17      if d[v]=-d[u]+1
           If width[v]<min(width[u],w[(u,v)])
               width[v]=min(width[u],w[(u,v)])
               Pi[v]=u
18A     Dequeue(Q)
18    color[u] ← black
```

The key changes are:

1. Maintain a new variable called width[u] which records the highest width path so far.

2. Whenever a new node is first encountered, both the width and distance is set.
3. If an alternate path is discovered (statement 17) then the width is checked and the path is updated.