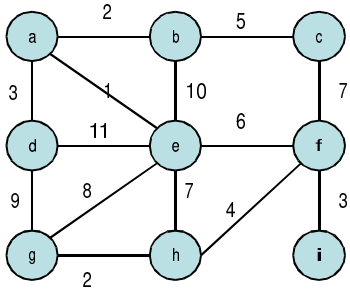
Clase Auxiliar CC30A 25/06/04

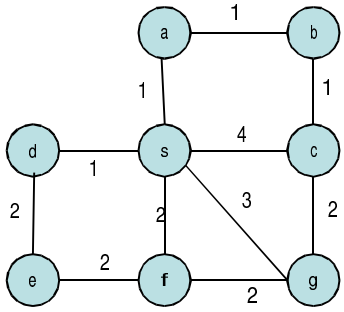
Problema 1

Aplique los algoritmos de Kruskal y Prim para obtener el árbol cobertor mínimo del siguiente grafo no dirigido.



Problema 2

Aplique el algoritmo de Dijkstra para obtener la ruta de distancia mínima entre el nodo s y todos los otros nodos del siguiente grafo:



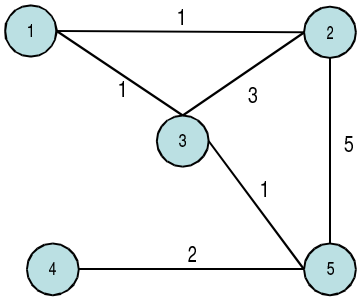
Problema 3

Una red de comunicaciones tiene enlaces (arcos), para cada uno de los cuales se conoce su confiabilidad, que es un número real entre 0 y 1, que indica la probabilidad de que funciones. La confiabilidad de una ruta dentro de la red es el producto de las confiabilidades de sus enlaces.

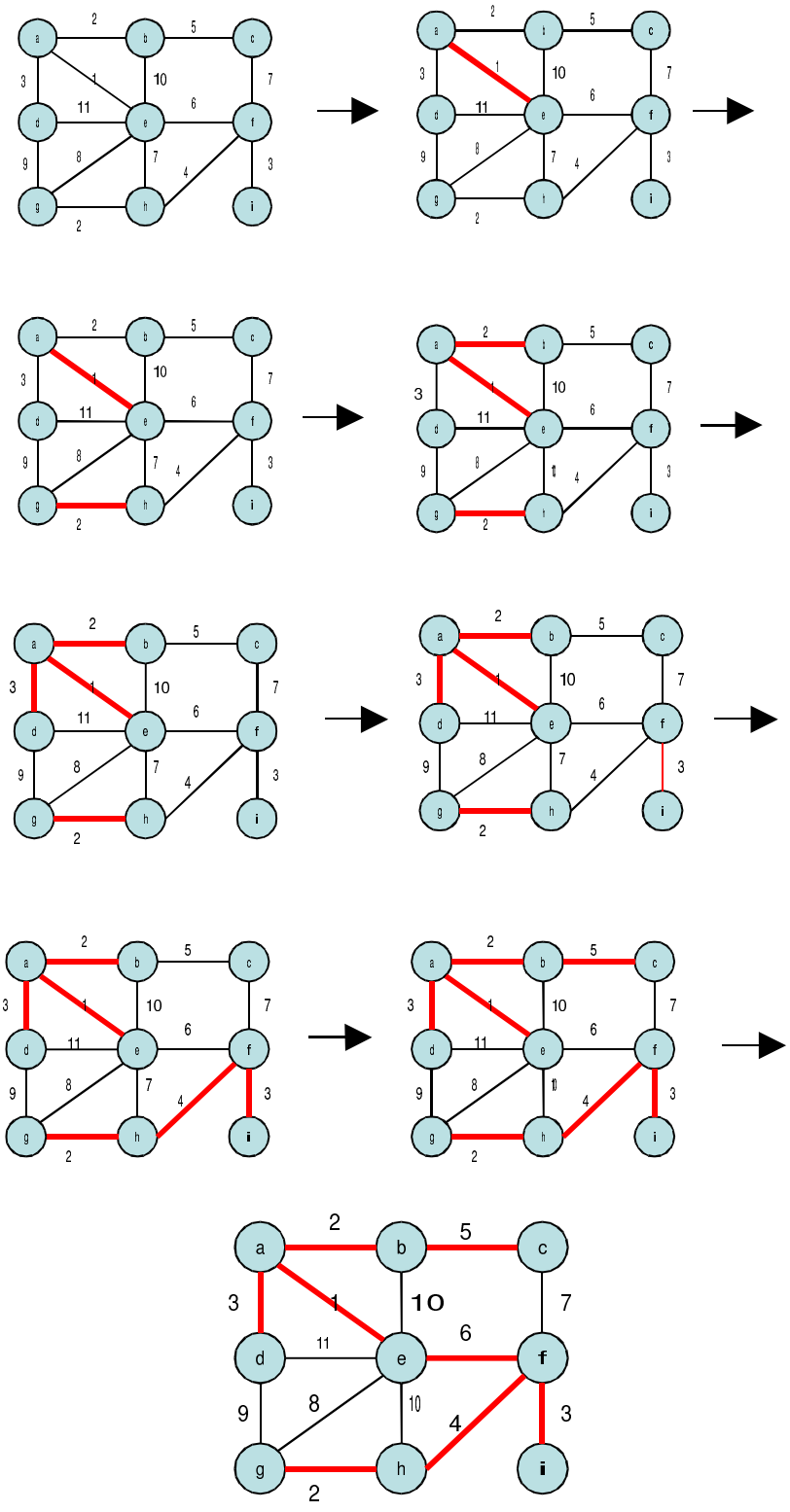
Describa cómo se podría utilizar algún algoritmo para grafos visto en clases para encontrar la ruta más confiable entre dos nodos dados.

Bonus Track

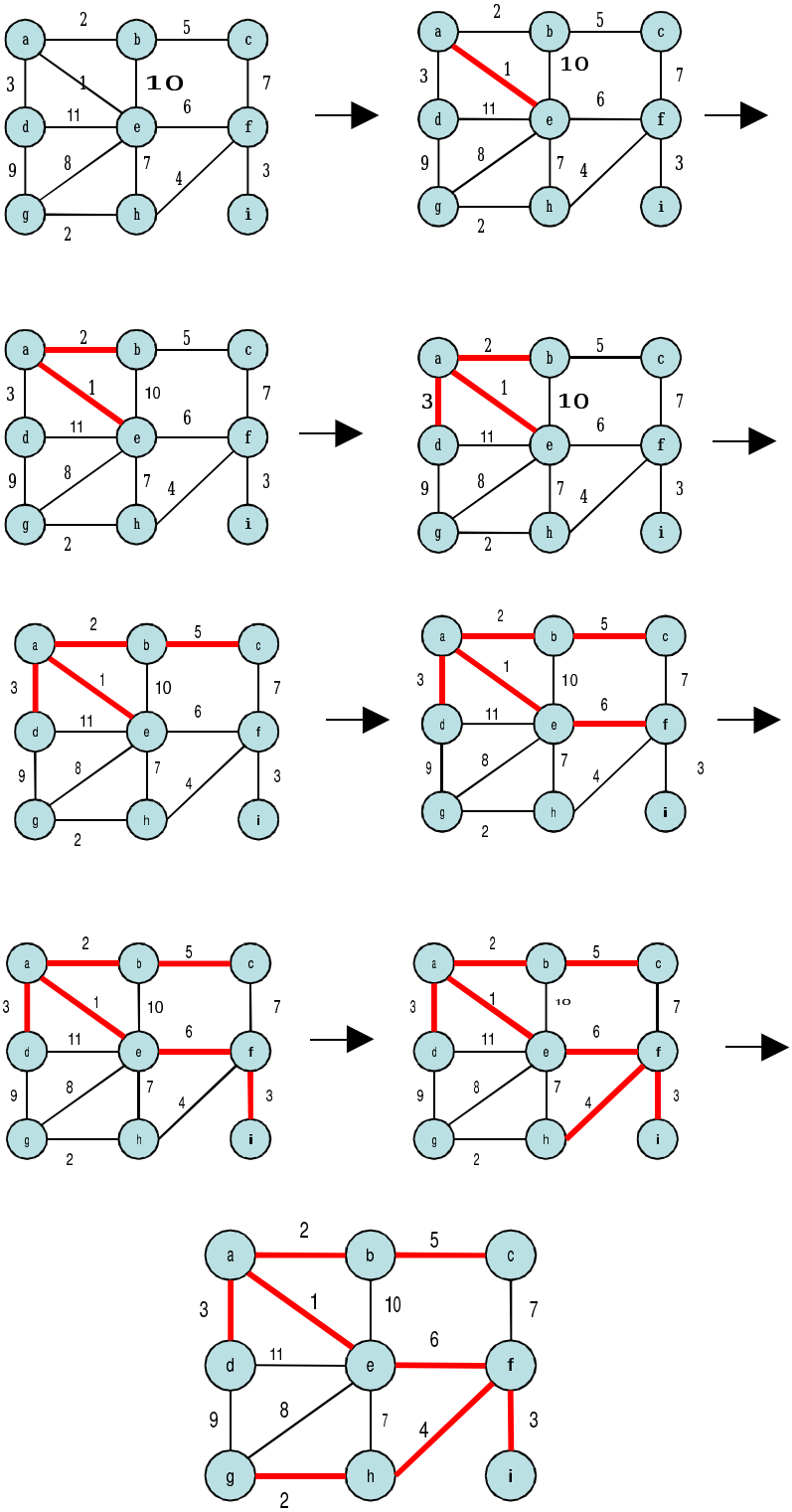
Aplique el algortmo de Floyd para encontrar todas las distancias mas cortas entre todos los nodos del siguiente grafo:



Solución P1:
a.- Kruskal:



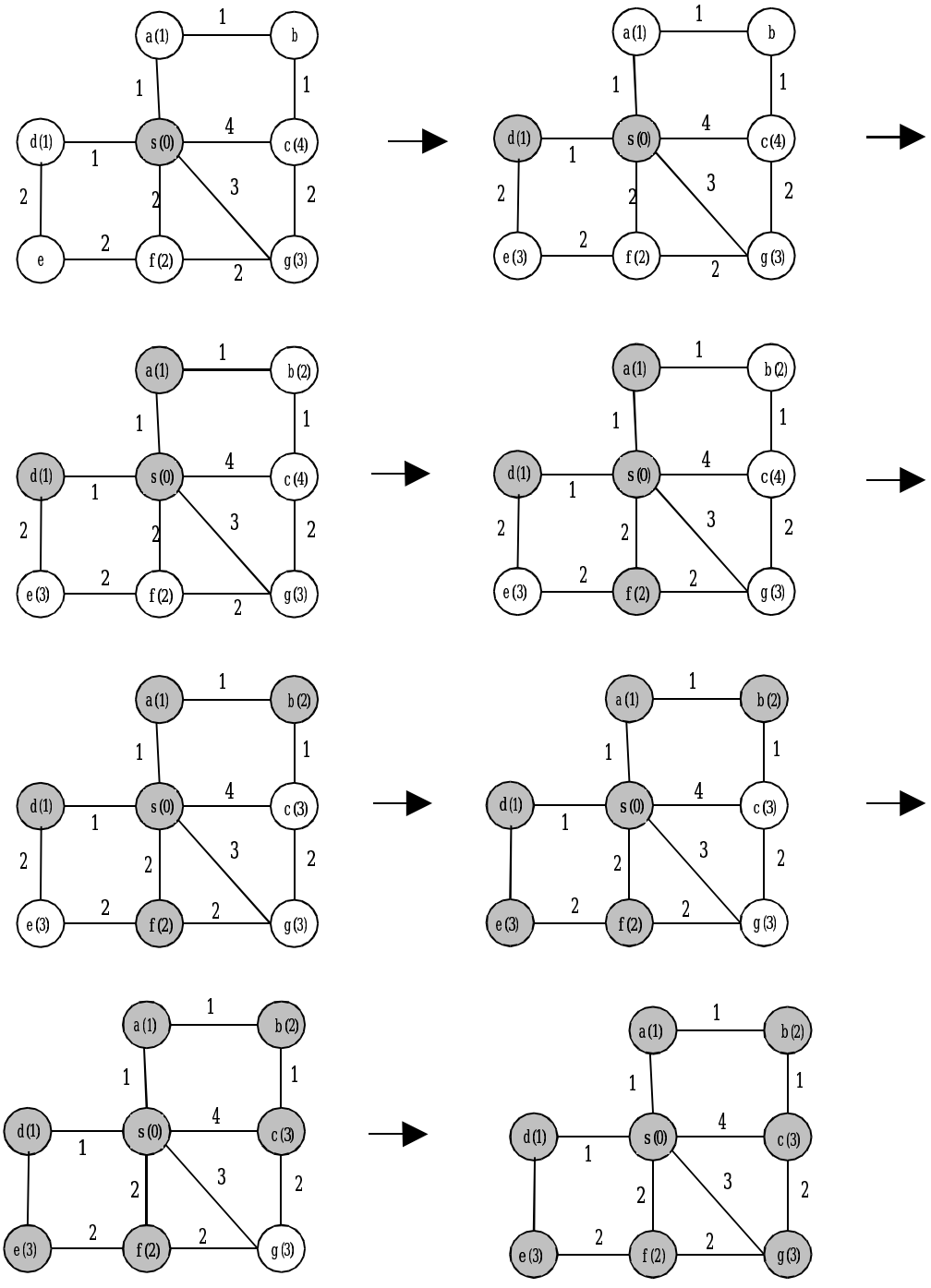
b.- Prim:



Solucion Problema 2

usaremos la siguiente notación:

nodos oscuros: ya alcanzados, dentro del conjunto A.
numero entre parentesis: distancia optima tentativa desde s hasta cada nodo. En el caso de los nodos oscuros, dicha distancia es completamente óptima (es decir, ya no es tentativa).



Solución Problema 3:

La ruta más confiable de un nodo a otro es aquella cuya probabilidad de que funcione es la mayor de todas las posibles. Para solucionar este problema se puede utilizar un algoritmo de distancia mínima modificado.
Veamos como quedaría el algoritmo de Dijkstra.
En vez de distancia mínima, buscamos la confiabilidad máxima. Además, para actualizar los valores de las confiabilidades (tabla D), en vez de sumar un nuevo costo, multiplicamos el valor actual por la nueva confiabilidad.

El algoritmo quedaría:

```
A={s};
D[s]=0;
D[v]=conf(s,v) para todo v en V-A; // 0 si el arco no existe
while(A!=V){
        Encontrar v en V-A tal que D[v] es máximo;
        Agregar v a A;
        for(todo w tal que (v,w) está en E)
            D[w]=max(D[w],D[v]*conf(v,w));
      }
```

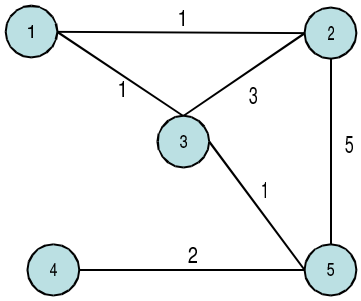
Una manera de modificar menos el algoritmo, sería calculando la misma ruta, pero considerando las probabilidades de fallar, la cual es 1-Pr(funcione). En este caso, queremos minimizar la probabilidad de que una ruta falle. El algoritmo quedaría:

```
A={s};
D[s]=0;
D[v]=falla(s,v) para todo v en V-A; // 1 si el arco no existe
while(A!=V){
        Encontrar v en V-A tal que D[v] es mínimo;
        Agregar v a A;
        for(todo w tal que (v,w) está en E)
            D[w]=min(D[w],D[v]*falla(v,w));
      }
```

Podemos utilizar el algoritmo original de Dijkstra sin ninguna modificación si es que en vez de almacenar en la tabla D la confiabilidad de un arco, guardamos –log(conf). En ese caso, las multiplicaciones se transforman en sumas. Se debe calcular el signo negativo para que la confiabilidad máxima se encuentre en la ruta de costo mínimo (Ej: log(1)=0 y log(0.1)=-1). La confiabilidad de un arco que no existe se debe definir como 0.

Solución Bonus Track:

Matriz de incidencia inicial:



	1	*2*	*3*	*4*	*5*
1	0	1	1	∞	∞
2	1	0	3	∞	5
3	1	3	0	∞	1
4	∞	∞	∞	0	2
5	∞	5	1	2	0

Ahora vemos si las distancias originales disminuyen al forzar pasar por el nodo 1. En este caso, sólo la distancia original entre los nodos 2 y 3 (que es 3), disminuye, pues d(2,1)+d (1,3)= 1+1=2:

	1	*2*	*3*	*4*	*5*
1	0	1	1	∞	∞
2	1	0	**2**	∞	5
3	1	**2**	0	∞	1
4	∞	∞	∞	0	2
5	∞	5	1	2	0

El mismo proceso, pasando por el nodo 2:

	1	*2*	*3*	*4*	*5*
1	0	1	1	∞	**6**
2	1	0	2	∞	5
3	1	2	0	∞	1
4	∞	∞	∞	0	2
5	**6**	5	1	2	0

Pasando por el nodo 3:

	1	*2*	*3*	*4*	*5*
1	0	1	1	∞	**2**
2	1	0	2	∞	**3**
3	1	2	0	∞	1
4	∞	∞	∞	0	2
5	**2**	**3**	1	2	0

Pasando por el nodo 4 no hace diferencias...veamos qué pasa si pasamos por el nodo 5:

	1	*2*	*3*	*4*	*5*
1	0	1	1	**4**	2
2	1	0	2	**5**	3
3	1	2	0	**3**	1
4	**4**	**5**	**3**	0	2
5	2	3	1	2	0

La tabla finalmente contiene las rutas mínimas entre cada par de nodos del grafo.