PROPIEDADES DEL DFS

```
#include<map>
#include<cstdio>
#include<vector>

using namespace std;

map<int,vector<int> > adj;

char color[1000];
int asc[1000],time,d[1000],f[1000];
int backedges,forwardedges,crossedges;
vector<int> toposort;

void DFS_VISIT(int u)
{
	color[u]='G';
	time++;
	d[u]=time;
	for(int i=0;i<adj[u].size();i++)
	{
		int v=adj[u][i];
		if(color[v]=='W')
			DFS_VISIT(v);
		else if(color[v]=='G')
			backedges++;
		else
		{
			if(d[u]<d[v])
				forwardedges++;
			else
				crossedges++;
		}
	}
	toposort.push_back(u);
	color[u]='B';
	time++;
	f[u]=time;
}

void DFS()
{
	for(map<int,vector<int> >::iterator it=adj.begin();it!=adj.end();it++)
	{
		color[it->first]='W';
		asc[it->first]=-1;
	}
	time=backedges=forwardedges=crossedges=0;
	for(map<int,vector<int> >::iterator it=adj.begin();it!=adj.end();it++)
	{
		if(color[it->first]=='W')
			DFS_VISIT(it->first);
	}
	return;
}

int main()
{
	int n;
```

```
        scanf("%d",&n);
        while(n--)
        {
                int u,v;
                scanf("%d %d",&u,&v);
                adj[u].push_back(v);
                vector<int> tmp;
                if(adj.find(v)==adj.end())
                        adj.insert(make_pair(v,tmp));
        }
        DFS();
        if(backedges==0)
        {
                printf("El grafo es aciclico. Ordenamiento topologico:\n");
                for(int i=toposort.size()-1;i>=0;i++)
                        printf("%d ",toposort[i]);
                printf("\n");
        }
        else
        {
                printf("El grafo tiene ciclos.\n");
                printf("Numero de back_edges: %d\n",backedges);
        }
        printf("Numero de fordward_edges: %d\n",forwardedges);
        printf("Numero de cross_edges: %d\n",crossedges);
        return 0;
}
```

BELLMAN-FORD

```
const int INF=1<<30;
int n,nedges,m[101][101];
int d[101],pi[101];

void initialize_single_source(int s)
{
        for(int i=0;i<n;i++)
        {
                d[i]=INF;
                pi[i]=-1;
        }
        d[s]=0;
        return;
}

void relax(int u,int v)
{
        if(d[v]>d[u]+m[u][v])
        {
                d[v]=d[u]+m[u][v];
                pi[v]=u;
        }
        return;
}

bool bellman_ford(int s)
{
        initialize_single_source(s);
        for(int i=0;i<nedges-1;i++)
```

```
                for(int i=0;i<n;i++)
                        for(int j=0;j<n;j++)
                        {
                                if(m[i][j]!=INF)
                                        relax(i,j);
                        }
        for(int u=0;u<n;u++)
                for(int v=0;v<n;v++)
                        if(m[u][v]!=INF && d[v]>d[u]+m[u][v])
                                return 0;               //Hay un ciclo negativo
        return 1;
}
```

DAG SHORTEST PATHS

```
const int INF=1<<30;
int n,nedges,m[101][101];
int d[101],pi[101];

char color[101];
vector<int> toposort;

void DFS_VISIT(int u)
{
        color[u]='G';
        for(int v=0;v<n;v++)
        {
                if(m[u][v]!=INF && color[v]=='W')
                        DFS_VISIT(v);
        }
        toposort.push_back(u);
        color[u]='B';
}

void DFS()
{
        for(int i=0;i<n;i++)
                color[i]='W';
        for(int i=0;i<n;i++)
        {
                if(color[i]=='W')
                        DFS_VISIT(i);
        }
        return;
}

void initialize_single_source(int s)
{
        for(int i=0;i<n;i++)
        {
                d[i]=INF;
                pi[i]=-1;
        }
        d[s]=0;
        return;
}

void relax(int u,int v)
{
        if(d[v]>d[u]+m[u][v])
        {
                d[v]=d[u]+m[u][v];
                pi[v]=u;
        }
        return;
}

void dag_shortest_path(int s)
{
        DFS();
        initialize_single_source(s);
        for(int u=toposort.size()-1;u>=0;u--)
                for(int v=0;v<n;v++)
                        if(m[u][v]!=INF)
                                relax(u,v);
}
```

PRIORITY QUEUE (Ejemplo)

```
#include <iostream>
#include <queue>

using namespace std;

struct Time {
        int h; // >= 0
        int m; // 0-59
        int s; // 0-59
};

class CompareTime {
public:
        bool operator()(Time& t1, Time& t2)
        {
                if (t1.h < t2.h) return true;
                if (t1.h == t2.h && t1.m < t2.m) return true;
                if (t1.h == t2.h && t1.m == t2.m && t1.s < t2.s) return true;
                return false;
        }
};

int main()
{
        priority_queue<Time, vector<Time>, CompareTime> pq;
        Time t[4] = { {3, 2, 40}, {3, 2, 26}, {5, 16, 13}, {5, 14, 20}};
        for (int i = 0; i < 4; ++i)
        pq.push(t[i]);
        while (! pq.empty()) {
        Time t2 = pq.top();
        cout << t2.h << " "  << t2.m << " " <<t2.s << endl;
        pq.pop();
        }
        return 0;
}
```