Dijkstra

```
#include <iostream>
#include <vector>
#include <utility>
#include <limits>
#include <algorithm>
#include <string>
#include <sstream>

struct Grafo
{
	// Matriz de adyacencia
	std::vector< std::vector<int> > mNodos;

	// Cargar una fila a la matriz
	void push(std::string s)
	{
		std::istringstream ss;
		int n;
		ss.str(s);

		mNodos.push_back(std::vector<int>());
		while (ss >> n)
			mNodos[mNodos.size()-1].push_back(n);
	}
};

// Predicado para el std::min_element
bool Predicado(std::pair<char, int> p1, std::pair<char, int> p2)
{
	return (p1.second < p2.second);
}

// Magia
int Dijkstra(Grafo& grafo, std::string &s)
{
	// Inicializar con infinito
	std::vector< std::pair<int, int> > v;
	for (int i = 0; i < grafo.mNodos.size(); i ++)
	{
		v.push_back(std::make_pair(i, std::numeric_limits<int>::max()));
	}

	// El primer nodo con cero
	v[0].second = 0;

	std::pair<int, int> u;
	std::vector< std::pair<int, int> >::iterator iter;

	// De donde viene cada nodo
	std::vector<int> back;
	back.resize(v.size());

	while (v.size() > 1)
	{
		iter = std::min_element(v.begin(), v.end(), Predicado);
		u = *iter;
		v.erase(iter);

		for (int i = 0 ; i < v.size() ; i ++)
		{
			// Si hay cero en la matriz es porque no hay arista entre los vértices
			if (grafo.mNodos[u.first][v[i].first] == 0)
				continue;

			if (u.second + grafo.mNodos[u.first][v[i].first] < v[i].second)
			{
				v[i].second = u.second + grafo.mNodos[u.first][v[i].first];

				// Actualizar de donde viene
				back[v[i].first] = u.first;
			}
		}
	}

	// Obtener el camino de vuelta
	int l = 0; // Longitud
	int k = grafo.mNodos.size()-1;
	s = static_cast<char>(k + 'a');
	std::ostringstream os;
	os << s;
	while (k != 0)
	{
		os << " <- " << static_cast<char>(back[k] + 'a');
		l += grafo.mNodos[k][back[k]];
		k = back[k];
	}
	s = os.str();

	return l;
}
```

```
int main()
{
	std::string s;

	// Cargar los grafos a partir de su matriz de adyacencia

	Grafo g;
	// Rosen Ejemplo 2 página 559
	g.push("0 4 2  0 0  0"); // a
	g.push("4 0 1  5 0  0"); // b
	g.push("2 1 0  8 10 0"); // ...
	g.push("0 5 8  0 2  6");
	g.push("0 0 10 2 0  3");
	g.push("0 0 0  6 3  0"); // f
	std::cout << Dijkstra(g, s) << " : " << s << std::endl;

	Grafo g1;
	//      b
	//     / \
	//   1/   \3
	//   /     \
	// a ---5--- c
	//   \      /
	//   2\    /2
	//     \  /
	//       d
	g1.push("0 1 5 2"); // a
	g1.push("1 0 3 0"); // b
	g1.push("5 3 0 2"); // c
	g1.push("2 0 2 0"); // d
	std::cout << Dijkstra(g1, s) << " : " << s << std::endl;

	Grafo g2;
	// Rosen ejercicio 3 página 562
	//     b---5---d---5---f
	//    /|      /|       |\
	//  4/ |     / |       | \7
	//  /  |    /  |       |  \
	// a   2   3   1       |   h
	//  \  |  /    |       |  /
	//  3\ | /     |       | /4
	//    \|/      |       |/
	//     c---6---e---5---g

	g2.push("0 4 3 0 0 0 0");
	g2.push("4 0 2 5 0 0 0");
	g2.push("3 2 0 3 6 0 0");
	g2.push("0 5 3 0 1 5 0");
	g2.push("0 0 6 1 0 0 5");
	g2.push("0 0 0 5 0 0 2");
	g2.push("0 0 0 0 5 2 0");
	g2.push("0 0 6 0 0 7 4"); // h
	std::cout << Dijkstra(g2, s) << " : " << s << std::endl;

	return 0;
}
```