MARTY MANOSTIJERAS

Para la realización del ejercicio, nos hemos decantado por el ejercicio del Domjudge 'Compras de la Semana', que se resuelve mediante back-tracking y puede haber varias podas.

Para comprobar el tiempo (en microsegundos) que tarda cada poda con cada ejemplo, en los códigos hay un temporizador que te muestra su tiempo de depuración y también hay un contador que hemos puesto para su comprobación experimental para ver cuántos nodos visita.

PODA 1

Para esta poda, el método que realizamos es crear un vector donde metemos los valores mínimos de los productos de todos los supermercados.

El cálculo de este vector tiene un coste O(n^2) y sus continuas comprobaciones durante el ejercicio tiene un coste O(n).

Es una poda muy poco eficiente, ya que es demasiado optimista y poda muy poco, ya que, a la hora de comparar los valores, solamente los comprueba con el mínimo de mi producto k+1 (en el caso de no haber llegado al final).

```
#include <iostream>
#include <time.h>
#include <fstream>
#include <climits>
#include <vector>
using namespace std;
void inicializaCompras(std::vector<std::vector<int>> &precio,
std::vector<int>&costesMinimosProd,const int&super, const int &prod)
{
	int k;
	for (int i = 0; i < super; i++)
	{
		for (int j = 0; j < prod; j++)
		{
			std::cin >> k;
			if (costesMinimosProd[j] > k) { costesMinimosProd[j] = k; }
			precio[i][j] = k;
		}
	}
}


bool esValido(const std::vector<int> &contador, const std::vector<bool>
marcaProd,const  int &i , const int &k)
{
	return  !marcaProd[k] && contador[i] < 3;
}


void comprasSemana(const std::vector<std::vector<int>> &precio, std::vector<bool>
&marcaProd, std::vector<int> &contador,
	int k , const int &super, const int &prod, int &suma, int &minimo, int
&cont, const std::vector<int> &costesMinimosProd)
```

```
{
	cont++;
	for (int i = 0; i < super; i++)
	{
		if (esValido(contador,marcaProd,i,k))
		{
			suma += precio[i][k];
			contador[i]++;
			marcaProd[k] = true;
			if (k == prod - 1)
			{
				if (suma < minimo)
				{
					minimo = suma;
				}
			}
			else {
				if (costesMinimosProd[k + 1] + suma < minimo)//poda
				{
					comprasSemana(precio, marcaProd, contador, k +
1, super, prod, suma, minimo, cont, costesMinimosProd);
				}
			}
			suma -= precio[i][k];
			contador[i]--;
			marcaProd[k] = false;


		}
	}
}


void resuelveCaso()
{
	int prod, super;
	int cont = 0;
	int suma = 0;
	int minimo = INT_MAX;
	clock_t t_ini, t_fin;
	double secs;

	std::cin >> super;
	std::cin >> prod;
	std::vector<int> contador(super, 0);
	std::vector<std::vector<int>> precio(super, std::vector<int>(prod));
	std::vector<bool> marcaProd(prod, false);
	std::vector<int>costesMinimosProd(prod, INT_MAX);
	inicializaCompras(precio, costesMinimosProd, super, prod);



	t_ini = clock();
	comprasSemana(precio, marcaProd, contador, 0, super, prod, suma, minimo,
cont, costesMinimosProd);
	t_fin = clock();
	secs = (double)(t_fin - t_ini) / CLOCKS_PER_SEC;
	cout << " tiempo " << secs*1000000.0 << endl;
	cout << minimo << "  " <<cont <<endl;
}
int main()
```

```
{
#ifndef DOMJUDGE
	std::ifstream in("casos.txt");
	auto cinbuf = std::cin.rdbuf(in.rdbuf());
#endif
	unsigned int n;
	std::cin >> n;
	// Resolvemos
	for (int i = 0; i < n; ++i) {
		resuelveCaso();
	}

	system("PAUSE");
	return 0;

}
```

PODA 2

Para esta poda nos hemos decantado por calcular un mínimo global de la matriz precio,

minimoTotal = min{precio[i,j] | 0 <= i <=n && 0 <=j <= n},

este valor nos sirve como cota inferior al precio de cada producto de los supermercados.

Con ello podemos aproximar el precio del resto de la solución con (prod – 1 – k)*minimoTotal.

Este cálculo de la aproximación es eficiente, ya que el mínimo global solo se calcula una vez (con un coste O(n^2)) y solo se mantiene una variable, pero su estimación sigue siendo muy poco afortunada.

```
#include <iostream>
#include <iomanip>
#include<math.h>
#include <fstream>
#include <vector>
#include <climits>
#include <algorithm>
using namespace std;

/*
 por supermercado a lo sumo compra 3 productos
 0 <= productos <= 3*supermercados y super <=20
*/
const int MAX_SUPER = 20;
const int MAX_PROD = 3 * MAX_SUPER;

bool esValido(const std::vector<int> &contador, const std::vector<bool>
&marcaProd, int i, int k)
{
	return  !marcaProd[k] && contador[i] < 3;
}


void comprasSemana(std::vector<int> &contador, const
std::vector<std::vector<int>> &precio, std::vector<bool> &marcaProd,
	int &suma, int &minimo, int k, int &cont, const std::vector<int>
&estimacion, int &precio_estimado, const int &minimoTotal)
```

```
{
	cont++;

	for (int i = 0; i < contador.size(); i++)
	{
		if (esValido(contador, marcaProd, i, k))
		{
			suma += precio[i][k];
			contador[i]++;
			marcaProd[k] = true;
			precio_estimado = suma +( minimoTotal * (marcaProd.size()-1 -
k));
			if (precio_estimado < minimo)
			{
				if (k == marcaProd.size() - 1)
				{
					minimo = precio_estimado;
				}
				else comprasSemana(contador, precio, marcaProd, suma,
minimo, k + 1, cont, estimacion, precio_estimado,minimoTotal);
			}

			suma -= precio[i][k];
			contador[i]--;
			marcaProd[k] = false;
		}
	}
}

void resuelveCaso()
{
	int prod, super;
	int suma = 0;
	int minimo = INT_MAX;
	std::cin >> super;
	std::cin >> prod;
	std::vector<int> contador(super, 0);
	std::vector<std::vector<int>> precio(super, std::vector<int>(prod));
	std::vector<bool> marcaProd(prod, false);
	std::vector<int> estimacion(prod);
	int precio_estimado = 0;
	int minimoTodo = INT_MAX;
	int cont = 0;
	for (int i = 0; i < super; i++) {
		for (int j = 0; j < prod; j++) {
			std::cin >> precio[i][j];
			if (precio[i][j] < minimoTodo)
			{
				minimoTodo = precio[i][j];
			}

		}
	}

	clock_t t_ini, t_fin;
	double secs;

	t_ini = clock();
	comprasSemana(contador, precio, marcaProd, suma, minimo, 0, cont,
estimacion,precio_estimado,minimoTodo);
	t_fin = clock();
```

```
	secs = (double)(t_fin - t_ini) / CLOCKS_PER_SEC;
	cout << " tiempo " << secs * 1000000.0 << endl;
	cout << minimo << "    " << cont << endl;
}
int main()
{
#ifndef DOMJUDGE
	std::ifstream in("casos.txt");
	auto cinbuf = std::cin.rdbuf(in.rdbuf());
#endif
	unsigned int n;
	std::cin >> n;

	// Resolvemos
	for (int i = 0; i < n; ++i) {
		resuelveCaso();
	}

	system("PAUSE");
	return 0;

}
```

PODA 3

Esta poda es la más razonable, consiste en tener calculado un mínimo por cada fila: tendremos calculado, para cada supermercado, el precio más barato.

costesMinimosProd[i] = min{precio[i,j] | 0 <= j <= n} , con ello podemos calcular la estimación del resto como:

- ∑ costesMinimosProd[i] (desde i = k+1 hasta n)

El vector costesMinimosProd se inicializa al principio con un coste O(n^2) pero ahora el cálculo de sus cotas es lineal.

Para que sea más eficiente, incluímos otro vector , llamado mejoresAcumulados, el cual también calcula al inicio(y por tanto, solo una vez) las sumas:

- mejoresAcumulados[k] = ∑ costesMinimosProd[i] (desde i = k+1 hasta n)

Con ello en el vector mejoresAcumulados guardamos los mejores valores para la compra y a la hora de realizar la poda, es mucho más estricto.

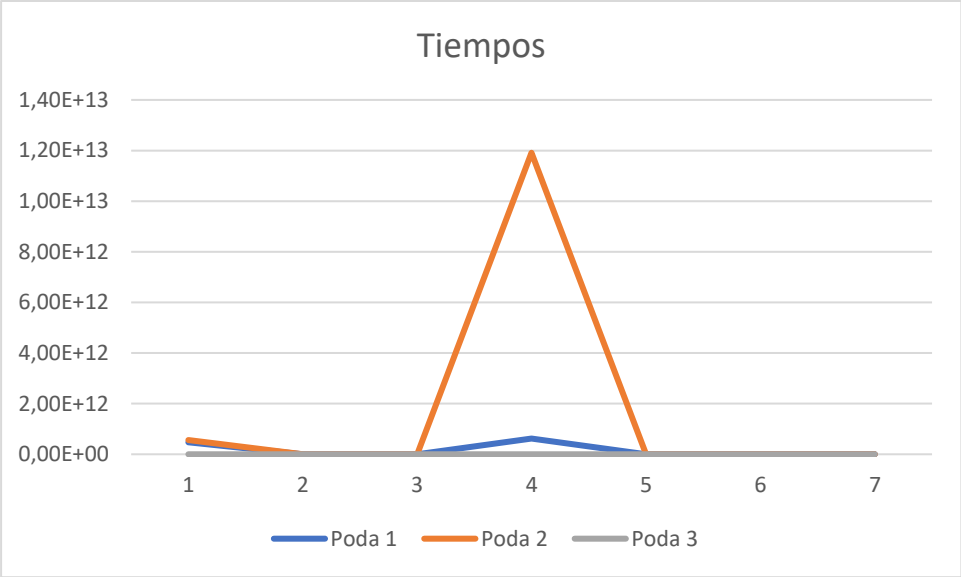
```
#include <iostream>
#include <fstream>
#include <vector>
#include <climits>
#include<time.h>
using namespace std;
```

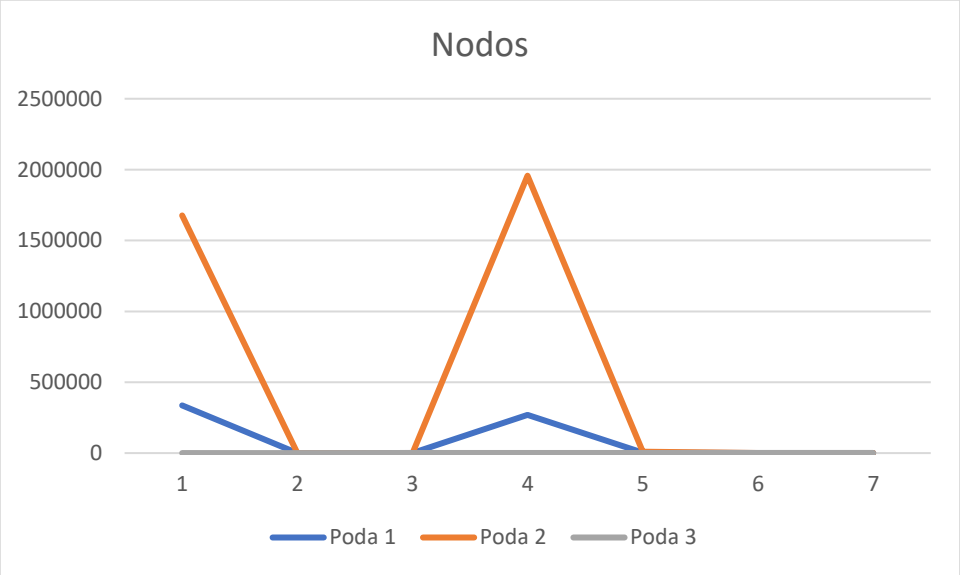
```
/*
 por supermercado a lo sumo compra 3 productos
 0 <= productos <= 3*supermercados y super <=20
*/

bool esValido(const std::vector<int> &contador, const std::vector<bool>
&marcaProd, int i, int k)
{
	return  !marcaProd[k] && contador[i] < 3;
}


void comprasSemana(std::vector<int> &contador, const
std::vector<std::vector<int>> &precio, std::vector<bool> &marcaProd,
std::vector<int> &costesMinimoProd,
	int &suma, int &minimo, int k, std::vector<int> &mejores, int &cont)
{
	cont++;
	for (int i = 0; i < contador.size(); i++)
	{
		if (esValido(contador, marcaProd, i, k))
		{
				suma += precio[i][k];
			contador[i]++;
			marcaProd[k] = true;
			if (k == marcaProd.size() - 1)
			{
				if (suma < minimo)
				{
					minimo = suma;
				}
			}
			else {
				if (mejores[k + 1] + suma < minimo)
				{
					comprasSemana(contador, precio,
marcaProd, costesMinimoProd, suma, minimo, k + 1,mejores,cont);
				}
			}
			suma -= precio[i][k];
			contador[i]--;
			marcaProd[k] = false;
		}
	}
}


void resuelveCaso()
{
	int prod, super;
	int suma = 0;
	int cont = 0;
	int minimo = INT_MAX;
	std::cin >> super;
	std::cin >> prod;
	std::vector<int> contador(super, 0);
	std::vector<std::vector<int>> precio(super, std::vector<int>(prod));
	std::vector<bool> marcaProd(prod, false);
	std::vector<int> costesMinimosProd(prod,INT_MAX);
	std::vector<int> mejoresAcumulados(prod);
```

```
	for (int i = 0; i < super; i++) {
		for (int j = 0; j < prod; j++) {
			std::cin >> precio[i][j];
			if (precio[i][j] < costesMinimosProd[j])
				costesMinimosProd[j] = precio[i][j];
		}
	}
	mejoresAcumulados.at(prod - 1) = costesMinimosProd.at(prod - 1);
	for (int i = prod - 2; i >=0; i--)
	{
		mejoresAcumulados[i] = mejoresAcumulados[i + 1] +
costesMinimosProd[i];
	}
	clock_t t_ini, t_fin;
	double secs;

	t_ini = clock();

	comprasSemana(contador, precio, marcaProd, costesMinimosProd, suma,
minimo, 0,mejoresAcumulados,cont);
	t_fin = clock();
	secs = (double)(t_fin - t_ini) / CLOCKS_PER_SEC;
	cout << " tiempo " << secs * 1000000.0 << endl;
	cout << minimo <<"   " << cont<< endl;
}
int main()
{
#ifndef DOMJUDGE
	std::ifstream in("casos.txt");
	auto cinbuf = std::cin.rdbuf(in.rdbuf());
#endif
	unsigned int n;
	std::cin >> n;

	// Resolvemos
	for (int i = 0; i < n; ++i) {
		resuelveCaso();
	}

	system("PAUSE");
	return 0;

}
```

GRÁFICAS

Cómo podemos comprobar, la poda número 2 es la que más tarda respecto a las otras, en ejecutarse. También es la que más nodos visita, como podemos ver en la gráfica de nodos. Entre la poda 1 y 3 parece que apenas hay diferencia en el tiempo, sin embargo, si nos fijamos en los nodos, la poda 3 es mucho más eficiente,





Casos de prueba utilizados

```
7
6 10
1820 510 370 1000 460 324 505 640 2030 409
2000 430 450 1110 606 290 530 670 2104 501
1760 502 395 1200 550 199 525 702 1830 550
2130 640 560 1307 735 450 600 720 2150 575
1143 455 505 1140 500 400 350 550 2030 399
1200 475 403 1002 560 350 502 640 2009 460
4 1
4020
3560
5540
3540
5 5
2 9 9 5 1
2 8 10 5 1
2 8 9 5 2
3 8 9 6 1
2 7 9 5 2
10 8
135 1478 1305 987 541 111 98 1338
150 1587 1255 958 600 115 95 1400
145 1578 1150 578 775 450 99 1395
155 1600 1000 550 900 350 150 1450
120 1578 1250 475 874 240 125 1333
111 1111 1300 500 900 300 120 1800
130 1358 1421 750 700 250 100 1630
130 1255 1200 600 350 222 80 1705
100 1800 2000 500 100 50 100 1400
105 1600 1425 350 500 350 85 1355
4 8
130 1358 1421 750 700 250 100 1630
130 1255 1200 600 350 222 80 1705
100 1800 2000 500 100 50 100 1400
105 1600 1425 350 500 350 85 1355
8 4
135 1478 1305 987
150 1587 1255 958
145 1578 1150 578
155 1600 1000 550
120 1578 1250 475
111 1111 1300 500
130 1358 1421 750
130 1255 1200 600
4 3
135 1478 1305
132 1477 1306
136 1475 1309
137 1476 1301
```

Importancia de datos de entrada

Las matrices más grandes tardan más en ser ejecutadas, debido a su tamaño, respecto a las más pequeñas. El orden en que tienen los datos también incide en el resultado del tiempo de ejecución, ya que no es lo mismo que los valores estén ordenados de menor a mayor coste que desordenados.