Minimum Spanning Tree (MST) - Ayrık Kümeler Kullanılarak Yapılan Kruskal'ın Algoritması

Burada ayrık setler kullanılarak yapılan Kruskal'ın MST'sinin kodu verilecektir. Bu algoritma O(MlogN) zamanda yapılmaktadır.

Tanım

Öncelikle kenarları azalmayan sırada sıralıyoruz. Bu işlemden sonra Ayrık Küme Birleşimi - Küme Oluşturma kullanarak toplam O(N) zamanda kenarları ağaca ekliyoruz. Sıralamayı takip ederek her bir kenarın ucundaki iki noktanın aynı ağaçta olup olmadığına her seferinde O(1) zamanda bakıyoruz(Toplamda O(M)). Son olarak iki ağacın birleşmesi de O(1) zamanda olduğundan algoritmamızın toplam karmaşıklığı O(MlogN + N + M) = O(MlogN).

Kodlama

Aşağıda randomlanmış Ayrık Küme Birleşimi kullanılmıştır.

```
vector<int> p (n);

int dsu_get (int v) { // noktanın hangi ağaca ait olduğu bulunurken ağaç aynı zamanda güncelleniyor
	return (v == p[v]) ? v : (p[v] = dsu_get (p[v]));
}

void dsu_unite (int a, int b) { // iki noktanın bağlı olduğu ağaçların birleştirilmesi
	a = dsu_get (a);
	b = dsu_get (b);
	if (rand() & 1)
		swap (a, b);
	if (a != b)
		p[a] = b;
}

// main kısmı

int m;
vector < pair < int, pair<int,int> > > g; // sırasıyla kenarın uzunluğu, 1. ve 2. nokta tutuluyor

// maliyeti hesaplama kısmı
int cost = 0;
vector < pair<int,int> > res;

sort (g.begin(), g.end()); // kenarları sırala
```

```
p.resize (n);
for (int i=0; i<n; ++i) // her bir nokta öncelikle ayrık bir küme
	p[i] = i;
for (int i=0; i<m; ++i) { // sırasıyla kenarlara bak
	int a = g[i].second.first,  b = g[i].second.second,  l = g[i].first;
	if (dsu_get(a) != dsu_get(b)) { // eğer iki ucu ayrı küme ise
		cost += l; // maliyeti ekle
		res.push_back (g[i].second); // kenarı sonuca al
		dsu_unite (a, b); // ve son olarak ağaçları birleştir
	}
}
```