BELLMAN-FORD

N node ve M edgeden oluşan bir graphta v nodeundan diğer bütün nodelara olan en kısa mesafeyi hesaplamak istiyoruz.

Dijkstra'nın Shortest Path algoritmasının aksine, bu algoritma negatif edge içeren graphlarda da kullanılabilir. Eğer graph negatif uzunluğa sahip bir döngü içeriyorsa bazı shortest pathler var olmayabilir(aslında shortest path -∞ olur). Bu algoritma, graphta negatif uzunluğa sahip döngü olup olmadığını bulmak için de kullanılabilir.

Algoritmanın Açıklaması

Şimdilik graphta negatif uzunlukta döngü olmadığını kabul ediyoruz. Negatif döngünün olduğu durum aşağıda ayrıca anlatılmıştır.

Algoritma çalışmayı bitirdikten sonra d [0...n-1] dizisi cevapları içeriyor olacak.Fakat başlangıçta d[v]=0 diğer elemanlar ise ∞ olarak atanmalıdır.

Bellman-Ford algoritması aşama aşama çalışır, her aşamada her edgei kullanmaya çalışır. Örneğin bir edge a numaralı node ile b numaralı node arasında olsun ve uzunluğu c olsun; eğer d[b], d[a]+c'den daha büyük ise b numaralı nodea ulaşmak için artık bu edgei kullanmak daha mantıklıdır.

Her edgei sırayla deneme aşamasını tam olarak n-1 kez yapmanın shortest pathleri hesaplamak için yeterli olacağı ispatlanmıştır(negatif uzunluğa sahip döngü olmadığı durumda). Ulaşılamayan bir nodeun d[] dizisindeki değeri ise ∞ olarak kalacaktır.

Kodlama

Bellman-Ford -diğer graph algoritmalarının aksine- graphı edgelerin listesi olarak tutmak için uygundur. Girdide n(node sayısı), m(edge sayısı), edge listesi ve başlangıc nodeunun numarası olan v verilir. Node numaraları 0'dan n-1'e kadardır.

INF değişkeni ∞'u temsil ediyor. INF'e öyle bir değer atamalıyız ki olabilecek herhangi bir pathin uzunluğundan daha büyük olsun. Bu sayede INF değişkenini gerçekten ∞ gibi davranabilsin.

```
struct edge {
	int a, b, cost;
};

int n, m, v;
vector<edge> e;
const int INF = 1000000000;

void solve() {
```

```
	vector<int> d (n, INF);
	d[v] = 0;
	for (int i=0; i<n-1; ++i)
		for (int j=0; j<m; ++j)
			if (d[e[j].a] < INF)
				d[e[j].b] = min (d[e[j].b], d[e[j].a] +
e[j].cost);
	// deneme amaciyla d[] dizisini yazdirabilirsiniz
}
```

if (d [e [j] .a] <INF) kontrolü graphtaki negatif edgelerden dolayı kullanılıyor. Bu kontrol olmadığı durumda negatif edgeler ∞ olan bir d[] değerini ∞-c ile güncelleyebilir, bu hatalı bir durumdur.

Gelişmiş Kodlama

Bu algoritma hızlandırılabilir. Genelde bütün shortest pathler n-1'den daha az sayıda adımda hesaplanır. Ve kalan adımlarda d[] dizisinde hiç güncelleme yapılmaz. Bu nedenle o anki adımda d[] dizisinde herhangi bir güncelleme yapılıp yapılmadığını belirten bir değişken daha tutacağız. Eğer bir adımda d[] dizsinde hiç bir değişiklik yapılmamışsa, o adımda algoritmayı sonlandırabiliriz(Bu optimizasyon algoritmanın asimptotik davranışını değiştirmez -yani bazı durumlar için n-1 adım da gerekli olabilir- lakin çoğu graphta algoritmayı oldukça hızlandırır -özellikle rastgele üretilmiş graphlarda-).

```
	void solve() {
	vector<int> d (n, INF);
	d[v] = 0;
	for (;;) {
		bool any = false;
		for (int j=0; j<m; ++j)
			if (d[e[j].a] < INF)
				if (d[e[j].b] > d[e[j].a] + e[j].cost) {
					d[e[j].b] = d[e[j].a] + e[j].cost;
					any = true;
				}
		if (!any)  break;
	}
	// deneme amaciyla d[] dizisini yazdirabilirsiniz
}
```

Yolları Çevirme

Bellman-Ford ile sadece shortest pathin uzunluğunu değil pathin kendisini de bulabiliriz.

Bunun için her nodeun kendi atasını(pathte o nodea gelmeden önceki son node) tuttuğu bir p[0...n-1] dizisi kullanmamız gerekiyor. Aslında herhangi bir a nodeuna olan shortest path p[a]'ya olan shortest pathe bir edge daha eklenmesiyle bulunur.

Bellman-Ford da aynı mantıkla çalışıyor, bir nodea olan shortest pathin hesaplanmış olduğunu varsayıp o nodedan çıkan edgelerle diğer nodelara olan shortest pathleri buluyor. Sonuç olarak önceki koda sadece p[] dizisini ekleyerek shortest pathin kendisini de bulabiliriz.

```
void solve() {
vector<int> d (n, INF);
d[v] = 0;
vector<int> p (n, -1);
for (;;) {
	bool any = false;
	for (int j=0; j<m; ++j)
		if (d[e[j].a] < INF)
			if (d[e[j].b] > d[e[j].a] + e[j].cost) {
				d[e[j].b] = d[e[j].a] + e[j].cost;
				p[e[j].b] = e[j].a;
				any = true;
			}
	if (!any)  break;
}

if (d[t] == INF)
	cout << "No path from " << v << " to " << t << ".";
else {
	vector<int> path;
	for (int cur=t; cur!=-1; cur=p[cur])
		path.push_back (cur);
	reverse (path.begin(), path.end());

	cout << "Path from " << v << " to " << t << ": ";
	for (size_t i=0; i<path.size(); ++i)
		cout << path[i] << ' ';
```

```
        }
}
```

t numaralı nodedan başlayarak p[] dizisini gezdiğimizde t'ye olan shortest pathi de terstten gezmiş oluyoruz.

Algoritmanın İspatı

Ulaşılamayan nodeların d[] dizisindeki değerleri ∞ olarak kalır.

i. adımda Bellman-Ford en fazla i-1 tane edge içeren bütün shortest pathleri bulmuş olur.

Başka bir deyişle, eğer v nodeundan a nodeuna olan shortest path k tane edge içeriyorsa algoritmanın k. adımında bu path kesinlikle bulunmuş olur.

İspat:
Shortest pathinde tam olarak k tane edge içeren bir a nodeu düşünün. Bu pathteki nodeları şu şekilde gösterelim:
$p_0=v, \ldots , p_k=a$
İlk adımdan önce $p_0=v$ bellidir. İlk adımda elimize bir (p_0,p_1) edgei gelecek, bu şekilde ilk adımdan sonra p_1 de bulunmuş olur. k adım sonra ise p_k'yi de bulmuş oluruz.

Söylenmesi gereken son şey ise herhagi bir shortest pathin en fazla n-1 tane edge içerebileceğidir. Bu nedenle algoritmayı n-1 adımda çalıştırmak yeterlidir.

Negatif Uzunluğa Sahip Döngü Durumu

Eğer negatif uzunlukta bir döngü varsa algoritma üzerinde birtakım değişikliklere gitmemiz gerekiyor. O döngüdeki her nodeun ve o döngüden ulaşılabilen her nodeun d[] dizisindeki değeri -∞ olmalıdır.

Bellman-Ford algoritması negatif uzunluklu döngüden ulaşılabilen bütün nodeların d[] dizisindeki değerlerini sonsuza kadar günceller. Eğer adım sayınısı n-1 ile sınırlandırmazsak algoritma sürekli faaliyet gösterecektir.

v nodeundan negatif uzunluğunda bir döngüye ulaşıp ulaşamayacağımız anlamak için Bellman-Ford'u n-1 değil n adımda çalıştırmamız gerekiyor. Eğer n. adımda d[] dizisinde bir tane bile güncelleme olursa bu demektir ki v nodeundan negatif uzunluklu döngüye ulaşılabiliyor. Eğer hiç güncelleme olmazsa ya öyle bir döngü yok ya da o döngüye v nodeundan ulaşılamıyor demektir.
Dahası, eğer böyle bir döngü varsa Bellman-Ford bu döngüdeki nodeları bulacak şekilde değiştirilebilri. n. adımda d[] dizisindeki değerini güncellediğimiz nodea x diyelim. x nodeu ya bu

döngünün içindedir ya da bu döngüden kendisine ulaşılabiliyordur. Döngüdeki nodelardan birini bulmak için x nodeunun n. atasını(bu nodea da y diyelim) bulmalıyız. y nodeunu bulduktan sonra elimize tekrar y nodeu gelene kadar y nodeunun atalarını gezmeliyiz(bunun çalışacağı kesindir çünkü döngünün içindeki nodelar için d[] dizisinde(dolayısıyla p[] dizisinde de) sürekli güncelleme olacaktır).

```
	void solve() {
	vector<int> d (n, INF);
	d[v] = 0;
	vector<int> p (n, -1);
	int x;
	for (int i=0; i<n; ++i) {
		x = -1;
		for (int j=0; j<m; ++j)
			if (d[e[j].a] < INF)
				if (d[e[j].b] > d[e[j].a] + e[j].cost) {
					d[e[j].b] = max (-INF, d[e[j].a] +
e[j].cost);
					p[e[j].b] = e[j].a;
					x = e[j].b;
				}
	}

	if (x == -1)
		cout << "No negative cycle from " << v;
	else {
		int y = x;
		for (int i=0; i<n; ++i)
			y = p[y];

		vector<int> path;
		for (int cur=y; ; cur=p[cur]) {
			path.push_back (cur);
			if (cur == y && path.size() > 1)  break;
		}
		reverse (path.begin(), path.end());

		cout << "Negative cycle: ";
		for (size_t i=0; i<path.size(); ++i)
			cout << path[i] << ' ';
```

```
        }
}
```

Bu döngüdeki nodeların d[] dizisindeki değerleri sürekli küçüleceği için integer sınırlarını aşabilirler. Bu durumdan kaçınmak için max fonksiyonu şu şekilde kullanılmıştır:

```
d[e[j].b] = max (-INF, d[e[j].a] + e[j].cost);
```

Yukarıdaki program v nodeundan ulaşılabilen bir negatif uzunluklu döngü olup olmadığını bulur. Bu algoritma graphta negatif uzunluklu bir döngü olup olmadığını bulmak için kullanılıyor fakat değiştirilebilir. Değiştirmek için başta bütün d[] değerlerini ∞ yerine 0 yapmalıyız(aynı anda her yerden olan shortest pathleri arıyormuşuz gibi). Negatif uzunluktaki döngülerin doğru bir şekilde tespiti bu durumdan etkilenmez.

Ayrıca “Finding the negative cycle in the graph” başlıklı konuları da araştırabilirsiniz.