Application of Bellman-Ford Algorithm Dijkstra's Algorithm

CSCI 3100

Example application of Shortest Path

In forex trading, one can exchange currency of one country to currency of another country. One example use EUR/USD where Euros are exchanged for US Dollars. Another example is USD/EUR where US Dollars are exchanged for Euros.

Let <C1>/<C2> = X be a currency pair quote

- C1 – base currency. Base currency is equal to 1 unit.
- C2 – counter currency. X is the amount of counter currency 1 unit of base currency can buy.

Example:

- USD/JPY = 113.37
- 1 US Dollar can buy 113.37 Japanese Yen

Suppose we have quotes for various currency pairs. We may be able to make some money on this.

Converting multiple currencies

USD/JPY = 113.37, JPY/EUR = 0.008, EUR/USD = 1.14

1 USD can buy 113.37 JPY.

113.37 JPY can buy 0.90696 EUR

0.90696 EUR can buy 1.0039 USD

So if we have 1000 USD, we can convert it to 1003.9 USD => Make $3.9

If we have 1000000 USD, we can convert it to 1003900 USD => Make $3900

This is called **Arbitrage Opportunity**

Arbitrage Opportunity

Suppose we are given:
- N currencies: $c_1, c_2, \ldots, c_N$
- N x N table of exchange rages R[x, y]: one unit of c_x can buy R[x, y] units of c_y

If there is a sequence of currencies: $c_{x1}, c_{x2}, \ldots, c_{xk}$ such that:
- $R[x_1, x_2]*R[x_2, x_3]*\ldots*R[x_k, x_1] > 1$
- Then we have an arbitrage opportunity

Represent each currency as a vertex.

Select edge weights such that:
- $R[x_1, x_2]*R[x_2, x_3]*\ldots*R[x_k, x_1] > 1$
- Then the cycle $(x_1, x_2, \ldots, x_k, x_1)$ is a negative weight cycle

Use Bellman-Ford algorithm to determine if a graph has a negative weight cycle

Selecting Edge Weights

Transform $R[x_1, x_2]*R[x_2, x_3]*\ldots*R[x_k, x_1] > 1$

To $f(R[x_1, x_2])+f(R[x_2, x_3])+\ldots+f(R[x_k, x_1]) > f(1)$ using logarithm

$\log(R[x_1, x_2]*R[x_2, x_3]*\ldots*R[x_k, x_1]) > \log(1) = 0$

$\log(R[x_1, x_2]) + \log(R[x_2, x_3]) \ldots+\log(R[x_k, x_1]) > 0$

$-\log(R[x_1, x_2]) - \log(R[x_2, x_3]) \ldots - \log(R[x_k, x_1]) < 0$

$\log(1/R[x_1, x_2]) + \log(1/R[x_2, x_3]) \ldots+\log(1/R[x_k, x_1]) < 0$

Weight of edge $(x_v, x_w) = \log(1/R[x_v, x_w])$

Finding negative weight cycle

Add dummy node s

Connect s to all vertices with an edge of weight 0

This ensures that every negative weight cycle is reachable from s

Run Bellman-Ford algorithm

If Bellman-Ford returns false, there is a negative weight cycle

This means there is an arbitrage opportunity

Example

Rates: R[v, w]

	X	Y	Z
X	1	8	1/4
Y	1/8	1	1/16
z	4	16	1

Edge Weights: lg(1/R[v, w])

	X	Y	Z
X	0	-3	2
Y	3	0	4
z	-2	-4	0

A. X->Y->Z->X forms a negative weight cycle, and thus presents an opportunity to make money
B. X->Y->Z->X forms a negative weight cycle; we would loose money on such an exchange
C. There is no opportunity to make money on any series of exchanges because R[v, w] = 1/R[w, v], for all currencies v and w.
D. X->Z->Y->X forms positive weight cycle, and thus presents an opportunity to make money
E. None of the above

Dijkstra's Algorithm: Shortest path in G, where G has non-negative edge weights

```
Dijkstra(G, s)
   for each v ∈ V
      d[v] = ∞;
      v.π = NULL
   d[s] = 0; S = ∅; Q = V;
   while (Q ≠ ∅)
      u = ExtractMin(Q);
      S = S U {u};
      for each v ∈ u->Adj[]
         if (d[v] > d[u]+w(u,v))
            d[v] = d[u]+w(u,v);
            v.π = u;
```

Relaxation Step

$O(|V|)$

$\to O(1)$

$\to \log(|V|)$

$|V|$

$\log|V|$

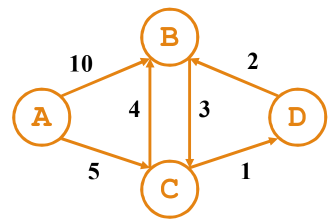
$\to total(|E|)$

Complexity:

$O(|V|\cdot\log|V| + |E|\cdot\log|V|) = \boxed{O((|V|+|E|)\cdot\log|V|)}$

Example: s = A.

On the first iteration of Dijkstra's algorithm, we'll extract vertex A from the priority queue and relax edges adjacent to A. What will be the estimates for each vertex in the priority queue at that time?



A. d[B] = 10, d[C] = 5, d[D] = 6

B. d[B] = 10, d[C] = 5, d[D] = ∞

C. d[B] = 9, d[C] = 5, d[D] = 6

D. d[A] = 0, d[B] = 10, d[C] = 5, d[D] = ∞

E. None of the above

Observations about Dijkstra's Algorithm

Greedy strategy
- Vertex with the smallest estimate is added to set S

One vertex is removed from the queue each time => queue will eventually become empty

Estimates of vertices in S don't change. To prove correctness, we need to show that d[x] = δ(s, x) when x is moved to S and s is the starting vertex.

Base case: true for the first vertex added to S. That vertex is s, d[s] = 0, and δ(s, s) = 0;

Inductive hypothesis: Suppose d[v] = δ(s, v), for all vertices in S. Show that when vertex u is moved to S, d[u] = δ(s, u)

Suppose $d[v] = \delta(s, v)$, for all vertices in S. Show that when vertex u is moved to S, $d[u] = \delta(s, u)$. How can we start such a proof?

A. Suppose $d[u] = \delta(s, u)$, when u is moved to S

B. Move vertex u to S and show that there is a shortest path from s to u in the subgraph formed by vertices of S.

C. Let $d[u] = d[s] + w(s, u)$, where $w(s, u)$ is the weight of edge from vertex s to vertex u.

D. Suppose $d[u] \neq \delta(s, u)$, when u is moved to S

E. Any of the above statements can be used to start this proof

Sketch of the correctness proof

Claim 1: There is a shortest path from s to u

Claim 2: Let p be the shortest path from s to u, where s is in S and u is in V-S. Let this path use an edge (x, y), where x is in S and y is in V-S. Then $d[y] = \delta(s, y)$.

Claim 3: $d[y] = \delta(s, y) \leq \delta(s, u) \leq d[u]$

Claim 4: Since vertex u was selected from the priority queue before vertex y, we know that $d[u] \leq d[y]$. But by claim 3, we have: $d[y] = \delta(s, y) \leq \delta(s, u) \leq d[u]$. This is only possible if $d[y] = \delta(s, y) = \delta(s, u) = d[u]$. This contradicts our assumption that $d[u] \neq \delta(s, u)$.

Claim 1 => Claim 2 => Claim 3 => Claim 4.

Recall the convergence property

If edge (u, v) is in the shortest path and d[u] = δ(s, u), then after relaxing edge (u, v), d[v] = δ(s, v). The convergence property can be used to prove:

A. Claim 1: There is a shortest path from s to u

B. Claim 2: Let p be the shortest path from s to u, where s is in S and u is in V-S. Let this path use an edge (x, y), where x is in S and y is in V-S. Then d[y] = δ(s, y).

C. Claim 3: d[y] = δ(s, y) ≤ δ(s, u) ≤ d[u]

Claim 1

There is a shortest path from s to u

$d[u] \neq \delta(s,u)$

If there is no path then

$\delta(s,u) = \infty$ and $d[u] = \infty$

$\Rightarrow \Leftarrow$

Claim 2:

Let p be the shortest path from s to u, where s is in S and u is in V-S. Let this path use an edge (x, y), where x is in S and y is in V-S. Then d[y] = δ(s, y).

$d[x] = \delta(s, x)$

When x moved to S, we relaxed (x, y). By convergence property $d[y] = \delta(s, y)$

Claim 3

$\underbrace{d[y] = \delta(s, y)}_{\text{Claim 2}} \le \delta(s, u) \le d[u]$