CS2040C Tut 11

Shortest Paths

Application Questions

Class Photo

Shortest Path

SSSP

APSP

Shortest Path Problems

Given a graph with weighted/unweighted edges, what is the **shortest path** from a vertex **X** to another vertex **Y**?

Shortest means the path with lowest “length”.

Shortest Path Problems

The definition of “length” can be vary between different problems

- Sum of edge weights of the path
- Number of edges (esp for unweighted)
- Sum of **vertex** weights
- *Product* of all edge weights
    - Only work if all weights are positive

Shortest Path Problems

In general, 2 types of shortest path problems:

Single Source Shortest Path (SSSP)

- Shortest path from a *single* starting vertex **S**, to *every other vertices*

All Pairs Shortest Path (APSP)

- Shortest path for *every pair* of vertices
- Essentially V times of SSSP

SSSP Algorithms

Bellman Ford

Dijkstra

Modified Dijkstra

SSSP Algorithms

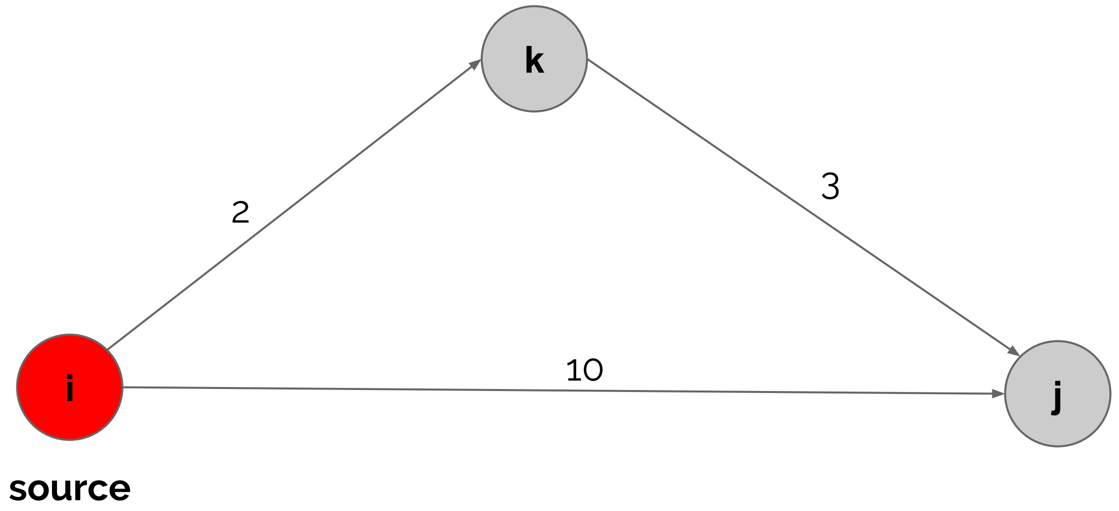
– Bellman Ford		$O(VE)$
– Dijkstra		$O((V+E) \log V)$
– Modified Dijkstra	≈	$O((V+E) \log V)$

Main Idea

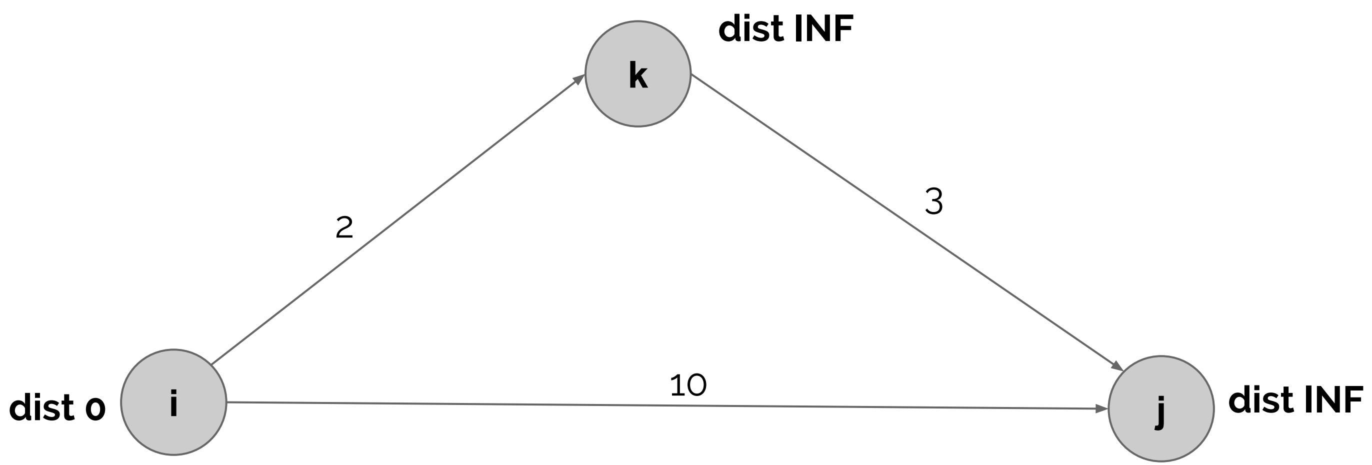
- Maintain a tentative “shortest distance”
- Iteratively attempts to “Relax” vertices
  - Just in different order!

What is Relax?

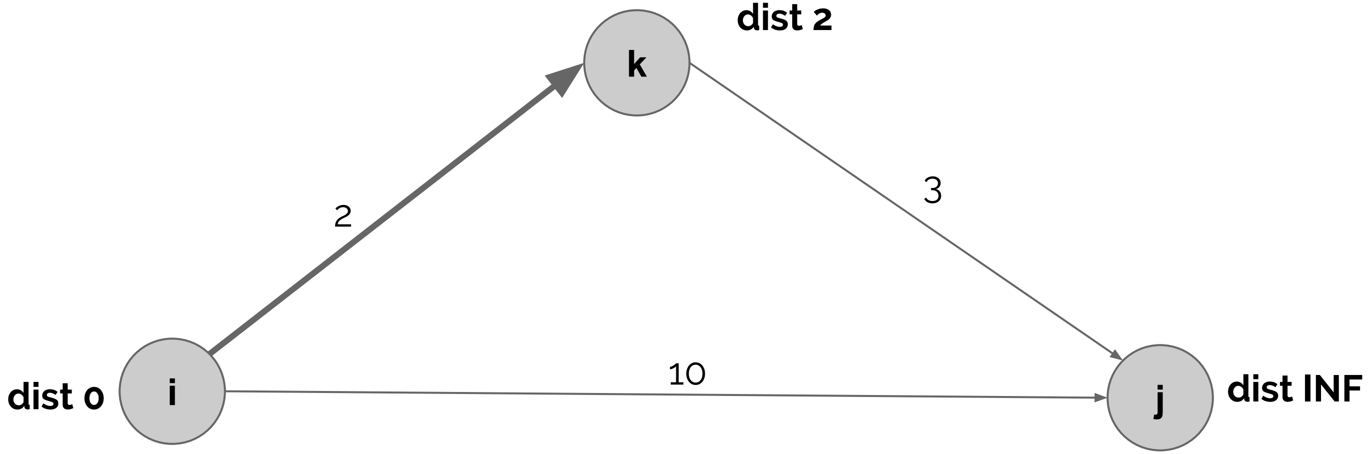
Relax



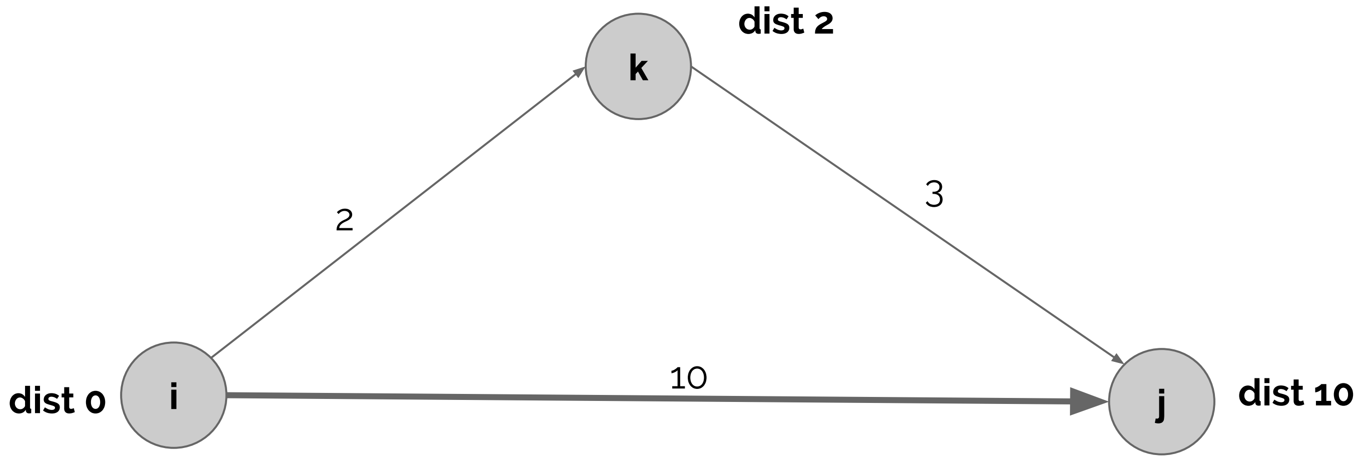
Relax



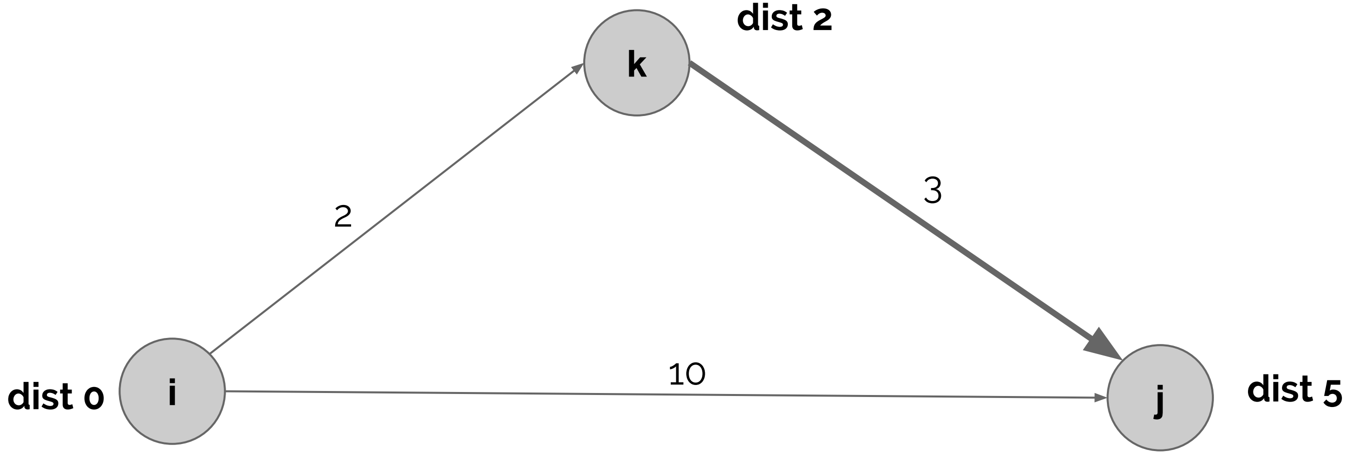
Relax



Relax

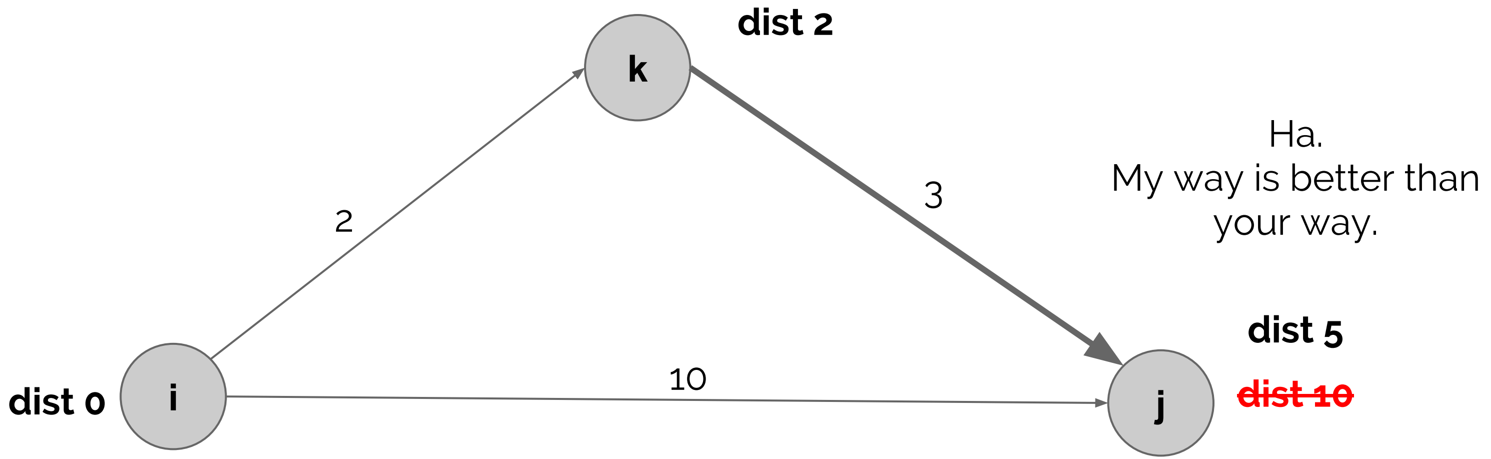


Relax



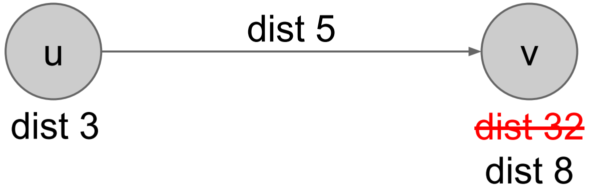
What is this *relax*?

Essentially traversing down **an edge** and updating the tentative ‘shortest distance’.



Bellman-Ford (Credits: NOI 2015 Dec Training Team)

```
dist[s] = 0; //dist all others = INF
for (int i = 0; i < N-1; i++){
     for (int j = 0; j < E; j++){
          dist[v] = min(dist[v], dist[u] + c);  //dir edge from u to v with c cost
     }
}
```



What is *infinity*?

What should be the value of infinity?

If we are using integers:

- INT_MAX (2^{31} - 1) ?
  - Will ***overflow***.
  - **INT_MAX + 1 = -2^{31}**

What is *infinity*?

What should be the value of infinity?

If we are using integers:

- Calculate an impossible value, based on the problem limits. (Eg: 10^9)
- Make sure **INF + max edge weight** fits into the data type
- Use *long long* if unsure!

What is *infinity?*

What should be the value of infinity?

Alternatively,

- We can use impossible ‘placeholder’ values such as -1.
- Need to handle such cases explicitly.
- Does not work with negative values.

(My preferred way, less bugs)

Dijkstra (Credits: NOI 2015 Dec Training Team)

```
const int MAXV = 100005;
typedef pair<int,int> pi;
vector<pi> adjList[MAXV];                                   // vertex, weight
// Note greater<pi> to reverse the order, can also use set<pi>
priority_queue<pi, vector<pi>, greater<pi> > pq;     // pair<distance, vertex>
int dist[MAXV], V, source;
bool visited[MAXV];

// initialize, -1 represents infinity, visited to false
for (int i = 0; i < V; ++i) dist[i] = -1, visited[i] = 0;
dist[source] = 0;           // set source to be dist 0
pq.push({0, source});       //insert into queue
```

Dijkstra (Credits: NOI 2015 Dec Training Team)

```
while(!pq.empty()){
    pi c = pq.top(); pq.pop();
    int curr = c.second, d = c.first;      // curr is vertex, d is dist
    if (visited[curr]) continue;           // lazy deletion method
    visited[curr] = 1;
    // for each pair<vertex, edge weight> in adj list of current vertex
    for (pi &i:adjList[curr]) {            // pair(vertex, weight)
        // condition for relax:
        //   1) tentative dist to i.first is infinity
        //   2) 'My way is better than your way'
        if(dist[i.first] == -1 || dist[i.first] > d + i.second){
            dist[i.first] = d + i.second;       //update distance
            pq.push({dist[i.first], i.first}); //push new dist into pq
        }
    }
}
```

Modified Dijkstra (Credits: NOI 2015 Dec Training Team)

```
const int MAXV = 100005;
typedef pair<int,int> pi;
vector<pi> adjList[MAXV];                               // vertex, weight
// Note greater<pi> to reverse the order
priority_queue<pi, vector<pi>, greater<pi> > pq;       // pair<distance, vertex>
int dist[MAXV], V, source;

// initialize, -1 represents infinity
for (int i = 0; i < V; ++i) dist[i] = -1;
dist[source] = 0;          // set source to be dist 0
pq.push({0, source});      //insert into queue
```

Modified Dijkstra (Credits: NOI 2015 Dec Training Team)

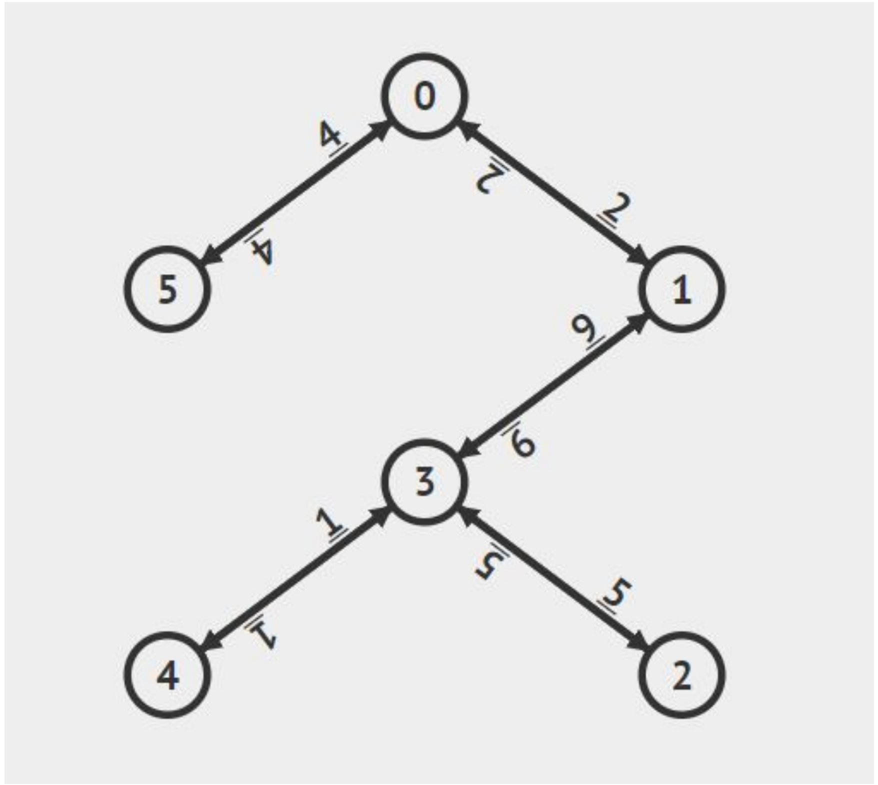
```
while(!pq.empty()){
    pi c = pq.top(); pq.pop();
    int curr = c.second, d = c.first;       // curr is vertex, d is dist
    if (d != dist[curr]) continue;           // modified dijkstra
    // for each pair<vertex, edge weight> in adj list of current vertex
    for (auto &i:adjList[curr]) {            // pair(vertex, weight)
        // condition for relax:
        //   1) tentative dist to i.first is infinity
        //   2) 'My way is better than your way'
        if(dist[i.first] == -1 || dist[i.first] > d + i.second){
            dist[i.first] = d + i.second;      //update distance
            pq.push({dist[i.first], i.first}); //push new dist into pq
        }
    }
}
```

SSSP UnvisuAlgo Quiz

Q1: Tree

Which algorithm can I use to solve SSSP from vertex 0?

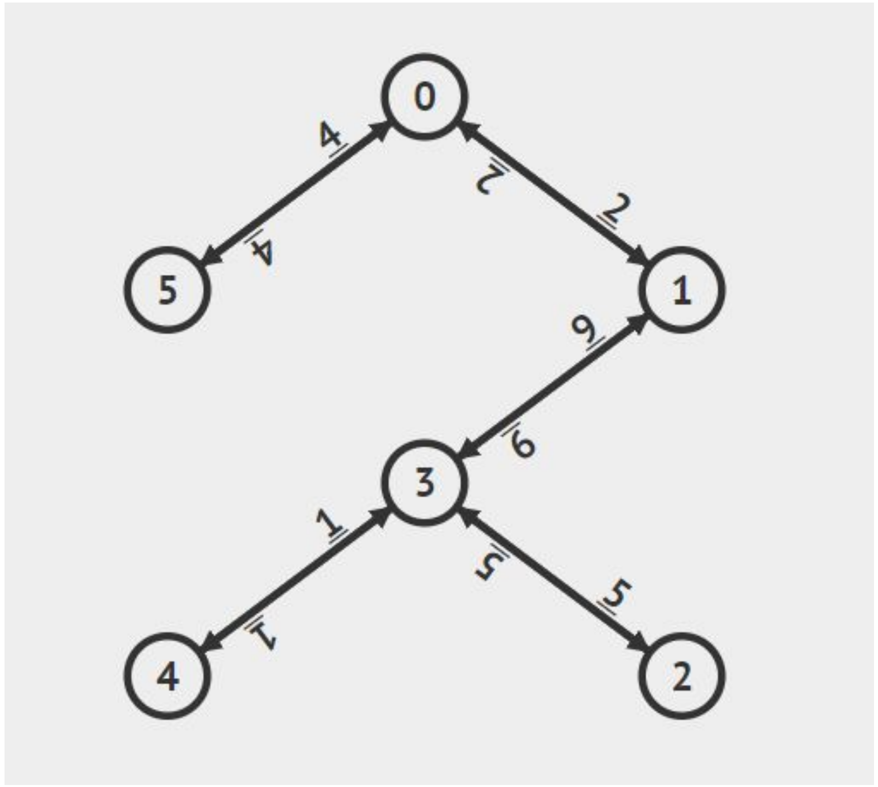
A. DFS
B. BFS
C. Bellman Ford
D. Dijkstra
E. Modified Dijkstra



Q1: Tree

Which algorithm can I use to solve SSSP from vertex 0?

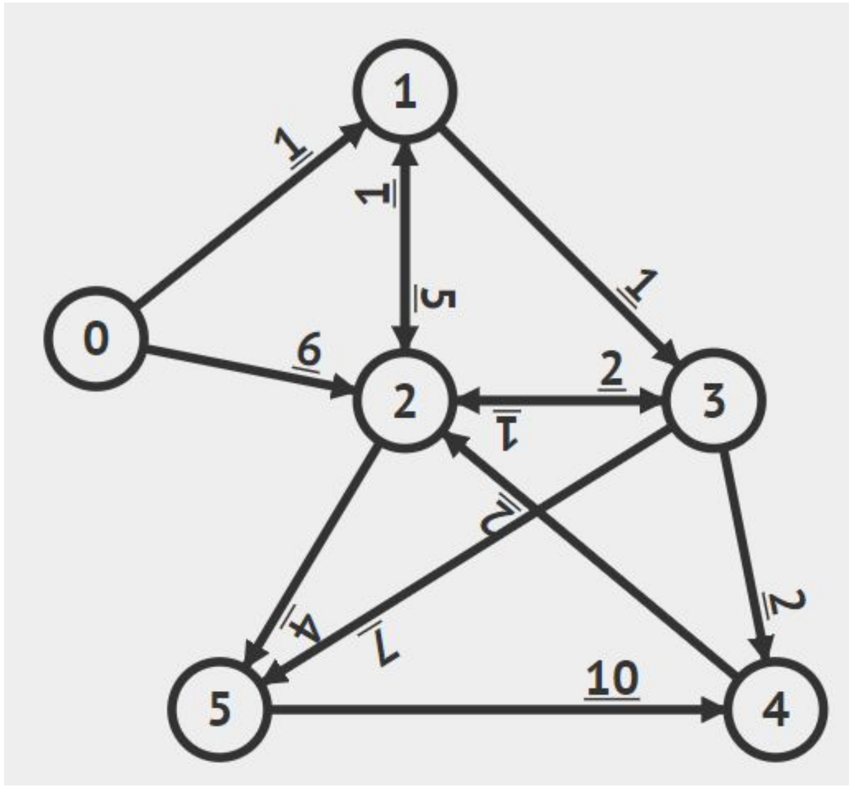
A. DFS (faster)
B. BFS (faster)
C. Bellman Ford
D. Dijkstra
E. Modified Dijkstra



Q2: Small General Graph

Which algorithm can I use to solve SSSP from vertex 0?

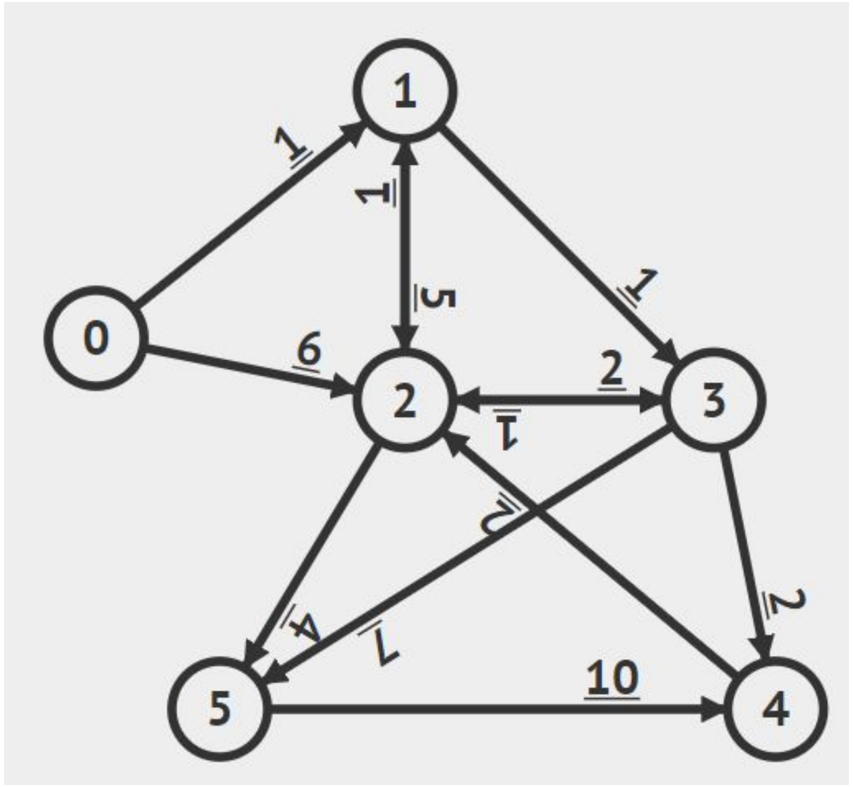
A. DFS
B. BFS
C. Bellman Ford
D. Dijkstra
E. Modified Dijkstra



Q2: Small General Graph

Which algorithm can I use to solve SSSP from vertex 0?

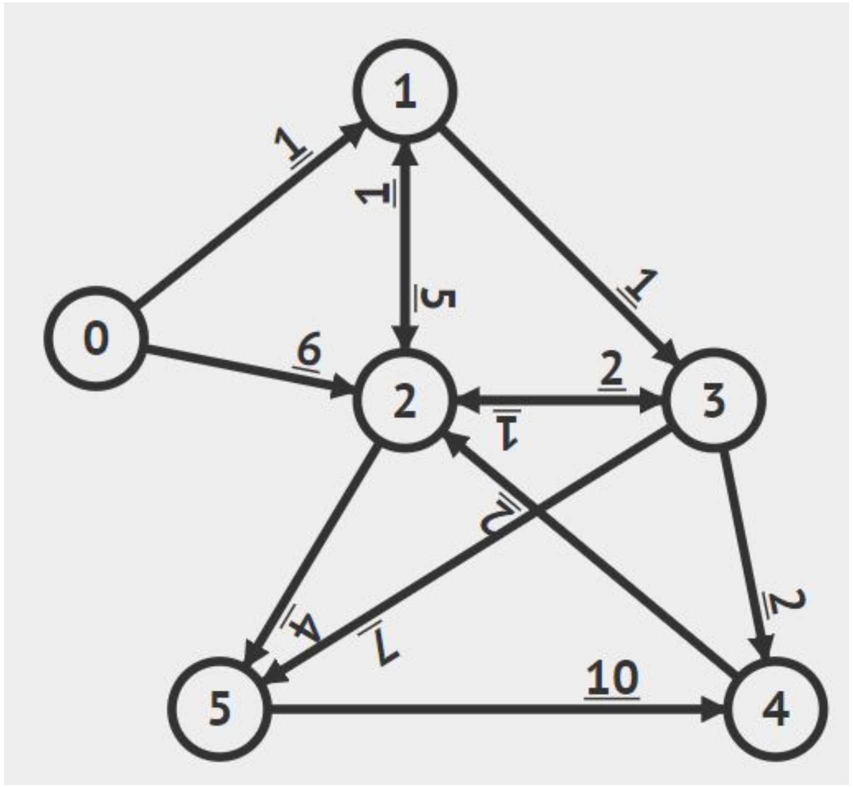
A. DFS
B. BFS
C. Bellman Ford
D. Dijkstra
E. Modified Dijkstra



Q3: General Graph

Which algo can I use to solve (many) SP **ending at vertex 2**?

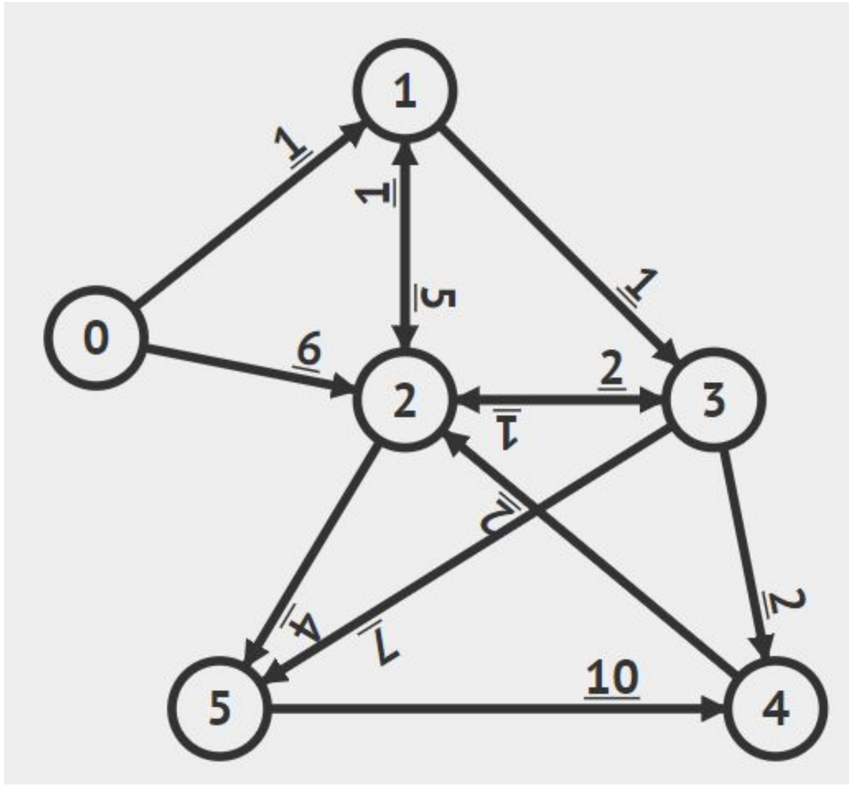
A. DFS
B. BFS
C. Bellman Ford
D. Dijkstra
E. Modified Dijkstra



Q3: General Graph

Which algo can I use to solve (many) SP **ending at vertex 2**?

Same, we can just **reverse the edge directions** and ‘start’ at vertex 2.



Q4: Large General Graph

Which algorithm can I use to solve SSSP from vertex 0?

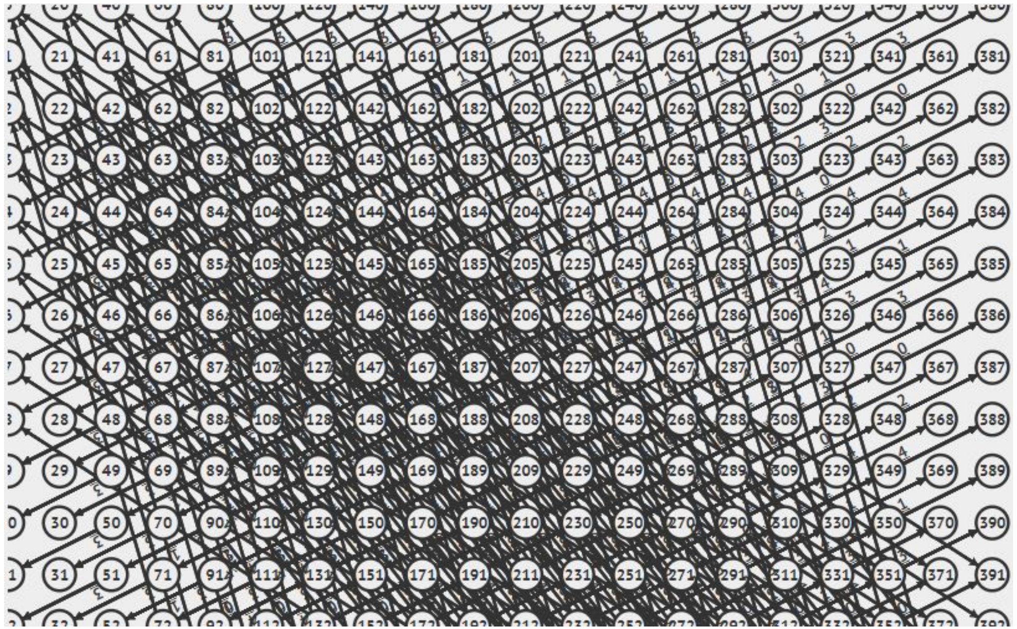
A. DFS
B. BFS
C. Bellman Ford
D. Dijkstra
E. Modified Dijkstra

Some large general graph :O

Q4: Large General Graph

Which algorithm can I use to solve SSSP from vertex 0?

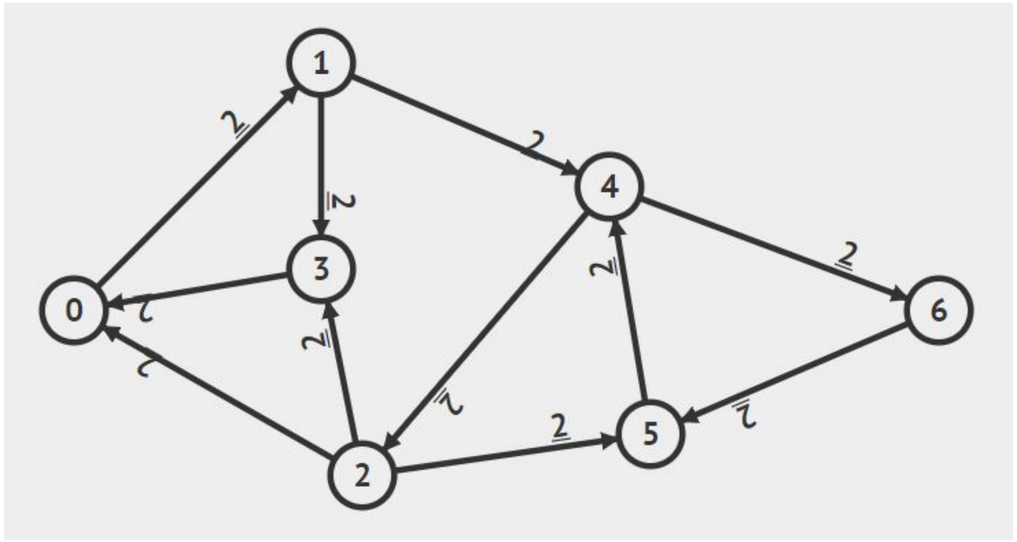
A. DFS
B. BFS
C. Bellman Ford
D. Dijkstra
E. Modified Dijkstra



Q5: Same Edge Weights

Which algorithm can I use to solve SSSP from vertex 0?

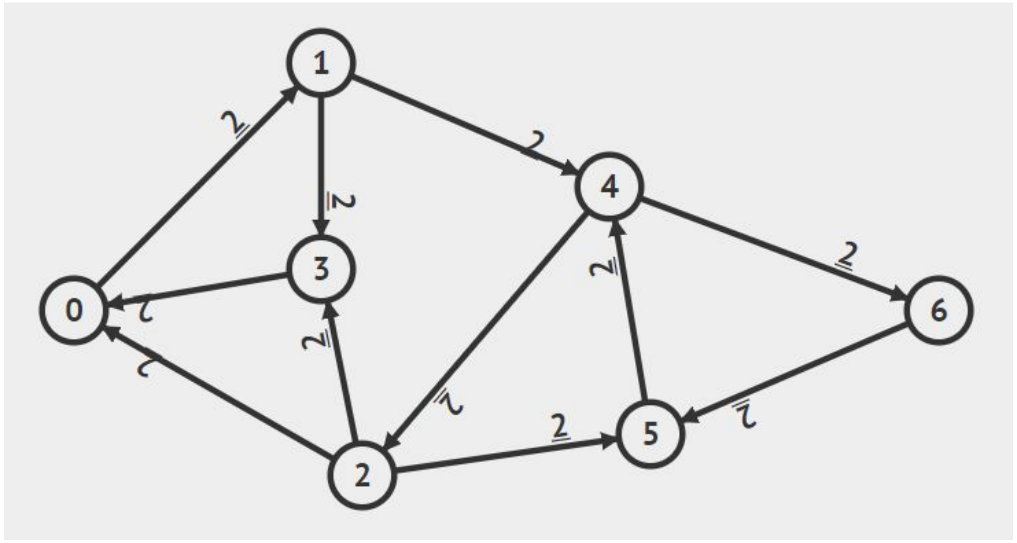
A. DFS
B. BFS
C. Bellman Ford
D. Dijkstra
E. Modified Dijkstra



Q5: Same Edge Weights

Which algorithm can I use to solve SSSP from vertex 0?

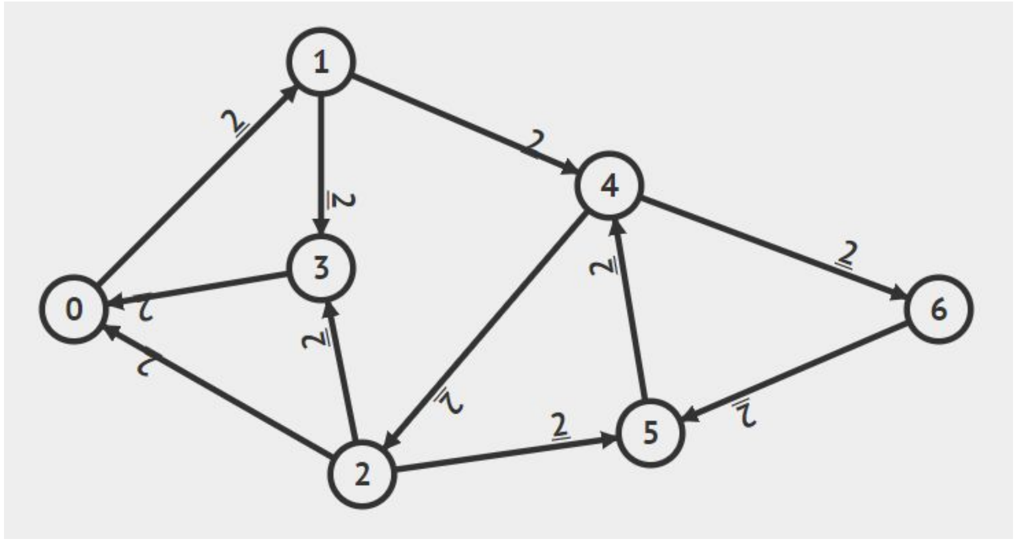
A. DFS
B. **BFS**
C. **Bellman Ford**
D. **Dijkstra**
E. **Modified Dijkstra**



Q5: Same Edge Weights

Which algorithm can I use to solve SSSP from vertex 0?

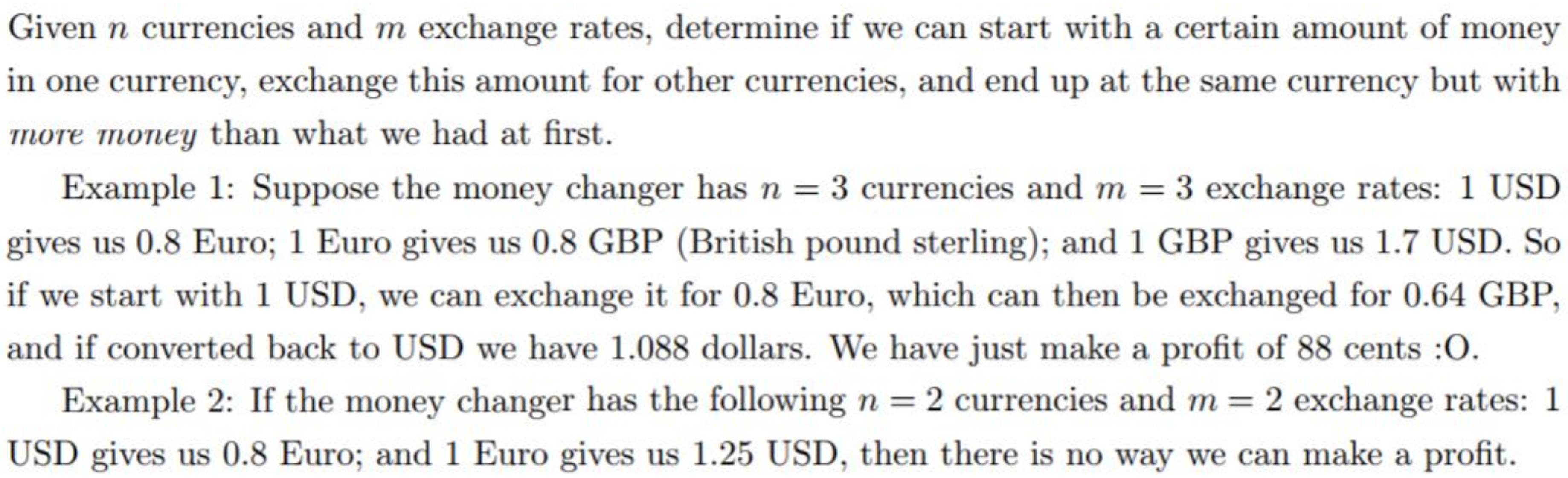
Since all edges have the same weight 2, we can treat the graph as **unweighted then, multiply answer by 2.**



Money Changer

CS2010 AY11/12 S1 Written Quiz 2

Money Changer

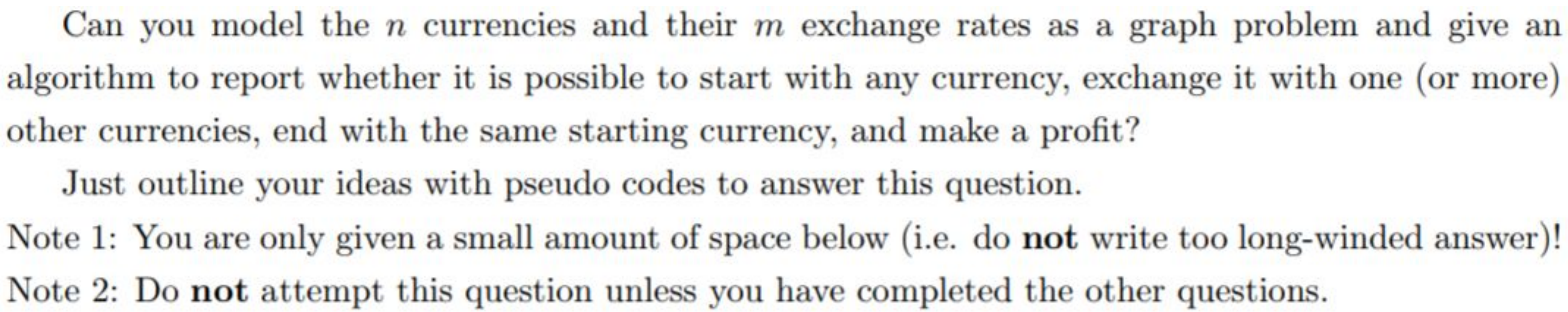


Given n currencies and m exchange rates, determine if we can start with a certain amount of money in one currency, exchange this amount for other currencies, and end up at the same currency but with *more money* than what we had at first.

Example 1: Suppose the money changer has $n = 3$ currencies and $m = 3$ exchange rates: 1 USD gives us 0.8 Euro; 1 Euro gives us 0.8 GBP (British pound sterling); and 1 GBP gives us 1.7 USD. So if we start with 1 USD, we can exchange it for 0.8 Euro, which can then be exchanged for 0.64 GBP, and if converted back to USD we have 1.088 dollars. We have just make a profit of 88 cents :O.

Example 2: If the money changer has the following $n = 2$ currencies and $m = 2$ exchange rates: 1 USD gives us 0.8 Euro; and 1 Euro gives us 1.25 USD, then there is no way we can make a profit.

Money Changer



Can you model the n currencies and their m exchange rates as a graph problem and give an algorithm to report whether it is possible to start with any currency, exchange it with one (or more) other currencies, end with the same starting currency, and make a profit?

Just outline your ideas with pseudo codes to answer this question.

Note 1: You are only given a small amount of space below (i.e. do **not** write too long-winded answer)!

Note 2: Do **not** attempt this question unless you have completed the other questions.

Graph Modelling

What are the vertices?

What are the edges?

Edge between? (direction?)

Edge weight?

Graph Modelling

What are the vertices? **Currency**

What are the edges? **Exchange Rate**

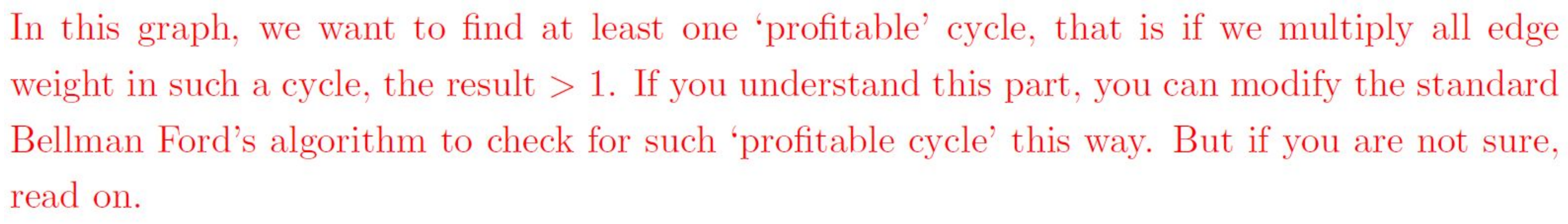
Draw an edge between **u** and **v** if there is an exchange rate to convert **u** to **v**.

Edge weight equals to the conversion rate.

Money Changer

But... where is the ‘profitable’ cycle?

A cycle where the ***product* of edge weights** is greater than 1.



In this graph, we want to find at least one ‘profitable’ cycle, that is if we multiply all edge weight in such a cycle, the result > 1. If you understand this part, you can modify the standard Bellman Ford’s algorithm to check for such ‘profitable cycle’ this way. But if you are not sure, read on.

Money Changer

Approach

We can modify Bellman Ford's algorithm.

dist[**v**] = max(dist[**v**], dist[**u**] * **rate**)

Start from starting currency:

dist[start] = *amount*; dist[others] = 0;

Money Changer

Approach

We can modify Bellman Ford’s algorithm.

dist[**v**] = max(dist[**v**], dist[**u**] * **rate**)

Check if **dist[start] > amount** at the end.

(We can actually assume *amount* = 1)

Money Changer

Alternative Approach

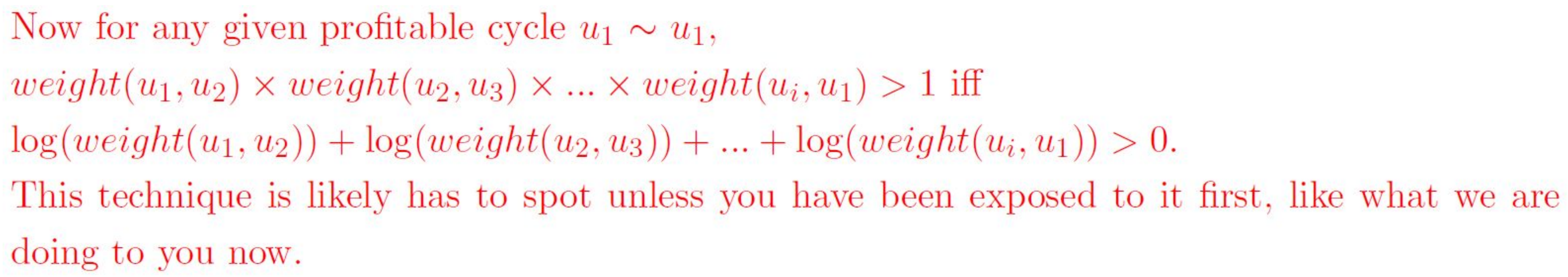
Convert multiplication into summation by using logarithm.

$$\mathbf{3 \times 4 \times 5 = 60}$$

$$\mathbf{\log(3) + \log(4) + \log(5) = \log(60)}$$

Money Changer

Alternative Approach



Now for any given profitable cycle $u_1 \sim u_1$,
$weight(u_1, u_2) \times weight(u_2, u_3) \times ... \times weight(u_i, u_1) > 1$ iff
$\log(weight(u_1, u_2)) + \log(weight(u_2, u_3)) + ... + \log(weight(u_i, u_1)) > 0$.
This technique is likely has to spot unless you have been exposed to it first, like what we are doing to you now.

Money Changer

Alternative Approach

- Now if we negate the logarithms, we have:
  $-\log(weight(u_1, u_2)) + -\log(weight(u_2, u_3)) + ... + -\log(weight(u_i, u_1) < 0.$
- So if we transform the exchange rate to the negative of its logarithm and use it to represent the edge weights, a 'profitable' cycle will means a negative cycle. From here, the answer becomes 'trivial'.

“Shortest” shortest path

Shortest path with least number of vertices

“Shortest” shortest path

Definition of “Shortest”

Which is the better path?

1. Distance = 17, Number of Cities = 4
2. Distance = 20, Number of Cities = 2

“Shortest” shortest path

Definition of “Shortest”

Which is the better path?

1. **Distance = 17, Number of Cities = 4**
2. Distance = 20, Number of Cities = 2

“Shortest” shortest path

Definition of “Shortest”

Which is the better path?

1. Distance = 17, Number of Cities = 4
2. Distance = **17**, Number of Cities = 2

“Shortest” shortest path

Definition of “Shortest”

Which is the better path?

1. Distance = 17, Number of Cities = 4
2. **Distance = 17, Number of Cities = 2**

“Shortest” shortest path

Definition of “Shortest”

In this priority:

1. Lower distance, *and if ties*
2. Lower number of cities

“Shortest” shortest path

Definition of “Shortest”

Essentially…

pair(distance, numCities)

Starting ‘distance’ is **pair(0, 1)**.

Each edge adds **pair(edge weight, 1)** to the ‘distance’.

“Shortest” shortest path

Definition of “Shortest”

Essentially…

pair(distance, numCities)

Priority Queue stores

pair(pair(distance, numCities), vertex)

Remember ‘distance’ should be first.

Shrinking Tunnels

CS2040C AY17/18 Sem 1 Final Exam

C4 C5

Main Difference

C4: All shrinking factor are **the same**

C5: Shrinking factor can differ

C4 solution

Approach

- Consider the graph as unweighted
- Run BFS, count the number of edges required from vertex **0** to vertex **N-1**
- If the result is **K**, the answer is $\mathbf{F}^{\mathbf{K}}$ where **F** is the shrinking factor

C4 Comments

Silly Mistake Cost You Marks

- The answer is $\mathbf{F^K}$, not **F*K**.
  - Common mistake
- How to compute $\mathbf{F^K}$?
  - **“^”** is bitwise xor in C++, **not power**!
  - **Can use O(K) loop** or *pow(base, exp)*

C4 Comments

Not-so-silly Mistake Also Cost You Marks

- C4 is ‘unweighted’, C5 is not.
- Best algorithm for C4 is BFS with O(V + E)
- Dijkstra solves C4 and C5, but **slower**!
  - O(V + E log V)
- We want the most efficient algorithm…
  - Dijkstra has -3 penalty
  - Bellman Ford has more penalty…

C5 solution

Approach

- A: Modify Dijkstra’s algorithm
- B: Modify the original graph to suit an unmodified Dijkstra’s algorithm

C5 solution

Approach A

- ‘dist[x]’ is the highest height you can be at vertex **x**
- Initial:
  - dist[0] = 1.0 “0”
  - dist[others] = 0 “infinity”

C5 solution

Approach A

- Relax condition
  - Edge from **u** → **v** with shrinking factor **f**

dist[**v**] = **max**(dist[v], dist[**u**]****f**)

C5 solution

Approach A

- Priority Queue
  - We want to process the highest height vertex first
  - Max-heap priority queue
  - That means we do **not** need to invert the default priority queue :O

C5 solution

Approach B

- Modify the original graph
- Shrinking Factor
  - Let the shrinking factor be **f**
  - Set the edge weight to be **-log(f)**
  - **log(f)** will be negative as 0.0 < f < 1.0
  - **-log(f)** will be positive

C5 solution

Approach B

$$\log(ab) = \log(a) + \log(b)$$

Maximizing product of shrinking factors is **maximising** sum of logarithms.

C5 solution

Approach B

$$\log(ab) = \log(a) + \log(b)$$

$$-\log(ab) = [-\log(a)] + [-\log(b)]$$

Maximizing product of shrinking factors is **minimizing** sum of negated logarithms.

C5 solution

Approach B

- Changing all edge weights to **-log(f)**
- Perform standard Dijkstra’s Algorithm
- Let the computed shortest path from 0 to N-1 be **A**
- Actual answer is $\mathbf{e^{-A}}$
  - Depending on which logarithm

C5 Comments

Common Mistakes

- Using min PQ when maximising distance
  - Approach A
- Cannot compute the answer back (i.e. e^{-A} ?)
  - Approach B

C5 Comments

- Approach A is more intuitive
  - Requires modifying standard Dijkstra’s algorithm
  - More things to write
- Approach B if you can *see* it
  - Might be difficult to spot without math background
  - Less writing, can cite ‘standard algorithm’

C5 Comments

- 75% got C4 and C5 correct
  - Excluding silly mistakes

- Designed to be an approachable question
  - Closely mimics “Money Changer” in Tut 10
  - It was ‘self-reading’ for that semester :P

Final Words

Feedback

Module Reviews

Final Exam

- Format similar to mid semester/past finals.
- Mainly application based.
- Tag/organise your notes.
  - "Well indexed notes == knowledge" (for open book)
- If you need reference to write the C++ code:
  - Don't, write in pseudocode first
  - Convert when you have time

Feedback

Reminder

Do complete feedback for CS2040C.

Your feedback will be valued :)

Thank You

- Sorry for always ending the tutorial late.
- Hope you have learnt a lot!

- Programming languages evolve.
- Applications of Data Structures/Graph are less likely to do so.

CS2040C ‘Alumni’ FB Group?

https://www.facebook.com/groups/NUS.CS2040C/

Members 267

Admins and Moderators 10

I will look through and approve.

Indicate your full name as per IVLE class roster :)

Questions?

Photo Taking