CS2040C Tutorial 11

Past Year Questions

PSA: Teaching feedback has opened. Please give the teaching team your feedbacks here.

Graph Traversal

CS2010 AY1516 Sem 1

C1

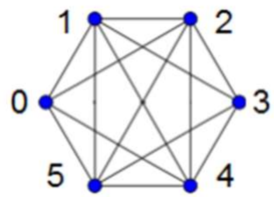


Figure 6: Near Complete Graph with $V = 6$ Vertices (One Edge (0, 3) is Missing)

You are given a near complete graph of **V** vertices numbered from **[0, V-1]** (5 < V < 100 000) with up to **k** edges missing ($0 \leq k \leq 7$).

Check if two vertices *i* and *j* are connected i.e. there is a path from *i* to *j*.

C1

Storage

Best way is to just store the Edge List of the $0 \leq k \leq 7$ missing edges.

If edge is not in the Edge List, then it exists.

C1

Naive Solution

BFS the entire graph.

Total complexity: $\mathbf{O(V+E) = O(V^2)}$

Graph traversal – Hint

Fact:

Consider a small graph $V = 4$, with up to $k = 2$ edges missing. Not possible for any vertex to be disconnected.

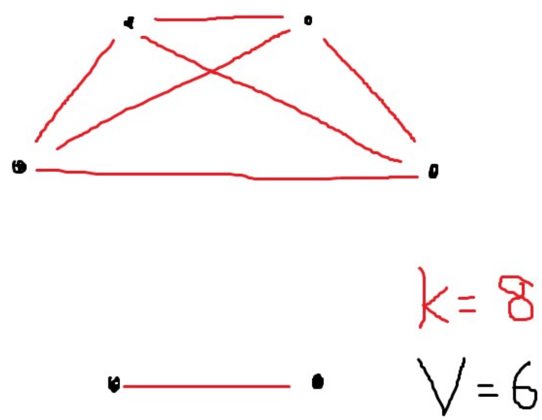
How about for $k = 7$, what V implies all vertices must be connected?

Graph traversal – Hint

Edge case:

Possible for disconnected vertex to have degree 1?

Come up with a counterexample:



C1

Observation

$$0 \leq k \leq 7$$

What happens if $V \geq 9$?

Can you disconnect any node from the graph taking away only k edges?

No!

As long as $k < V - 1$ the graph is connected.

C1

Observation

$$0 \leq k \leq 7$$

So we only need to look at $5 < V \leq 8$.

Graph traversal – Solution

- For $V \geq 9$, simply return true.
- For $6 \leq V \leq 8$, either:
  - Perform BFS/DFS, traversing along edges that are not present in the list of missing edges, $O(V^2)$
  - Check if either vertex has degree 0, i.e. appears $V - 1$ times in the edge list, $O(k)$

- Effectively constant time algorithms

Printing Integers

CS1020E AY1617 Sem 1

B1

You are given an array **A** consisting of **N** *distinct* integers, **N** up to 10,000,000.

Given **L** and **R**, print all numbers in **A** that are in the range **[L, R]**, in *sorted order*. However, **L + 10000 ≥ R**.

All numbers fit in 32-bit signed data type.

B1

Observation

L + 10000 ≥ R implies **R - L ≤ 10000**

⇒ There are at most 10001 numbers in this range.

⇒ Values in **A** are distinct → at most 10001 numbers to output

B1

Better Solution

Loop through the entire array **A** and extract numbers that lie in **[L, R]**.

STL Sort those numbers separately and print.

Total complexity: **O(N + (R-L) log (R-L))**

B1

Even Better Solution

Loop through the entire array **A** and extract numbers that lie in **[L, R]**.

Since R-L is at most 10000,

We can use **Counting Sort** to sort those numbers separately and print.

Total complexity: **O(N + (R-L))**

B1

Implementation

Since the numbers are *distinct*, the counting sort does not even need to count them.

⇒ It either appears once or not at all.

Boolean array to indicate whether a number is present or not will do.

Interesting Variants

CS2040C AY1819 Sem 2

B2

You are probably familiar with the following standard declarations of C++ STL stack, queue, and priority queue of integers:

```
stack<int> s;
queue<int> q;
priority_queue<int> pq;
```

B2

All these three C++ STL classes are actually **wrappers of underlying containers**, which by default is a C++ STL `deque` for stack and queue or a C++ STL `vector` for priority queue. But there is a way to ask `stack`, `queue`, or `priority_queue` to use a **specific** underlying container, e.g.

B2

```
stack<int, deque<int>> s; // is 100% identical to stack<int> s
// stack uses back(), push_back(), and pop_back() of its container
queue<int, deque<int>> q; // is 100% identical to queue<int> q
// queue uses back(), front(), push_back(), and pop_front() of its container
priority_queue<int, vector<int>> pq; // is 100% identical to priority_queue<int> pq
// priority_queue uses front(), push_back(), pop_back(), and [] of its container
```

B2

Knowing this, you are now interested to know if the following will work:

```
stack<int, vector<int>> s1; // put your comment at box 1 below
stack<int, list<int>> s2; // put your comment at box 2 below
queue<int, vector<int>> q1; // put your comment at box 3 below
queue<int, list<int>> q2; // put your comment at box 4 below
priority_queue<int, deque<int>> pq1; // put your comment at box 5 below
priority_queue<int, list<int>> pq2; // put your comment at box 6 below
```

B2

Please predict and elaborate on what will happen to `s1`, `s2`, `q1`, `q2`, `pq1`, and `pq2` above. The options are:

A. Cannot compile,
B. Becomes (slightly) slower
C. Gives wrong behavior
D. No significant change
E. Becomes (slightly) faster

B2

```
stack<int, vector<int>> s1;
```

Yes, can compile.

stack	vector
`top()`	`back()`
`push()`	`push_back()`
`pop()`	`pop_back()`

E. “slightly faster “ since `vector` is less bloated than `deque`.

B2

```
stack<int, list<int>> s2;
```

Yes, can compile.

stack	list
top()	back()
push()	push_back()
pop()	pop_back()

D. “no significant change“ since `list` and `deque` are more or less comparable.

B2

```
queue<int, vector<int>> q1;
```

Cannot compile.

No `pop_front()` operation.

Even if have, would be O(n) since have to close the gap.

B2

```
queue<int, list<int>> s2;
```

Yes, can compile.

`queue`	`list`
`front()`	`front()`
`push()`	`push_back()`
`pop()`	`pop_front()`

D. “no significant change“ since `list` and `deque` are more or less comparable.

B2

```
priority_queue<int, deque<int>> pq1;
```

Yes, can compile.

`priority_queue`	`deque`
`top()`	`front()`
`push()`	`push_back()`
`pop()`	`pop_back()`
`[]` #random access	`[]` #random access

B. “slightly slower” since `vector` is less bloated than `deque`.

B2

```
priority_queue<int, list<int>> pq2;
```

Cannot compile.

No `[]` random access operation.

Isomorphic BSTs

CS2040C AY1718 Sem 1

C1

Warning

Isomorphic in this question is actually defined as **“exactly the same”**.

Two “*isomorphic*” BSTs should be equivalent.

C1

For up to full 15 marks, your solution must work for $1 \leq N \leq 20$.

For up to partial 8 marks, your solution must work for $1 \leq N \leq 3$ (think carefully).

Observation

There are only 3! + 2! + 1! possible inputs to consider for $N \leq 3$.

Only 9 … try all of them. Abt 1 mark each XD lol

C1

Approach

For full score, since **N** is still small, we can choose to **simulate the binary search tree** for both insert sequences.

Then, compare if the resultant tree is the same.

C1

Issue

How do we check if the two trees are the same?

- In-order traversal?
- Pre-order traversal?
- Post-order traversal?

C1

Issue

How do we check if the two trees are the same?

- In-order traversal?
- Pre-order traversal?
- Post-order traversal?

In-order + either Pre/Post order traversal uniquely defines any tree.

C1

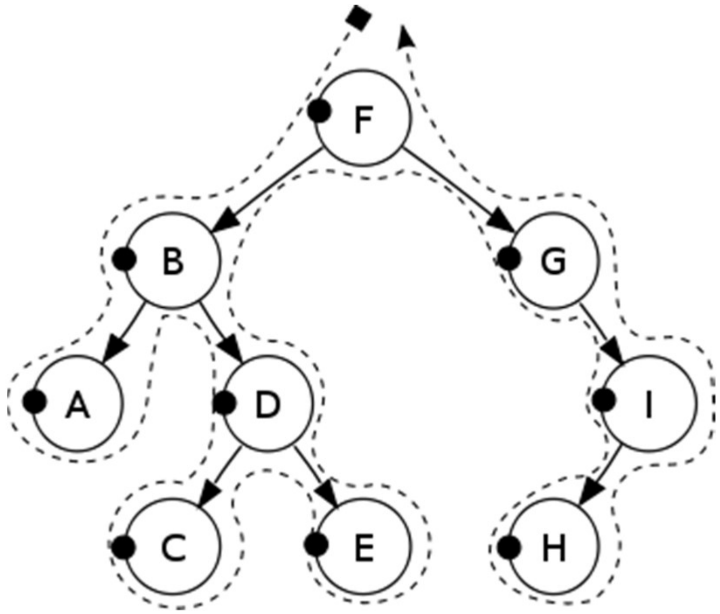
Approach 1

- The in-order traversal of a BST is just the **sorted order** of the vertices.
- We just need one more traversal.
- **DFS** gives us the pre-order traversal of the tree.

(Optional, self-read)

Pre-order Traversal

[Credits: SG IOI Training 2017, Wikipedia]



F B A D C E G I H

C1

Approach 1

- Since both insert sequences are permutations of 1 to N, the sorted order and hence the in-order is the same.
- We are just left to check if the **pre-order traversal is the same**.

C1

Implementation (Approach 1)

- Construct the two BST. $O(N^2)$
- Perform DFS to obtain the pre-order traversal of the two BST. $O(N)$
- Loop and compare if the traversals are the same. If yes → 'isomorphic' $O(N)$

C1

Similar Alternative

- Checking the topological sort order of both BST will also work.
  - Equivalent to reverse post-order traversal of the tree.

C1

Comments

- Such insights might be hard to think of (and prove) in the exam.
- There is actually a more *“straightforward”* way to check if two BSTs are equivalent.
  - However, harder to code.

C1

Approach 2 - Equality check

Two BSTs are equal if the **structure** and the **value** of vertices are equal.

- Structure is the same
- Value is the same

C1

Observation

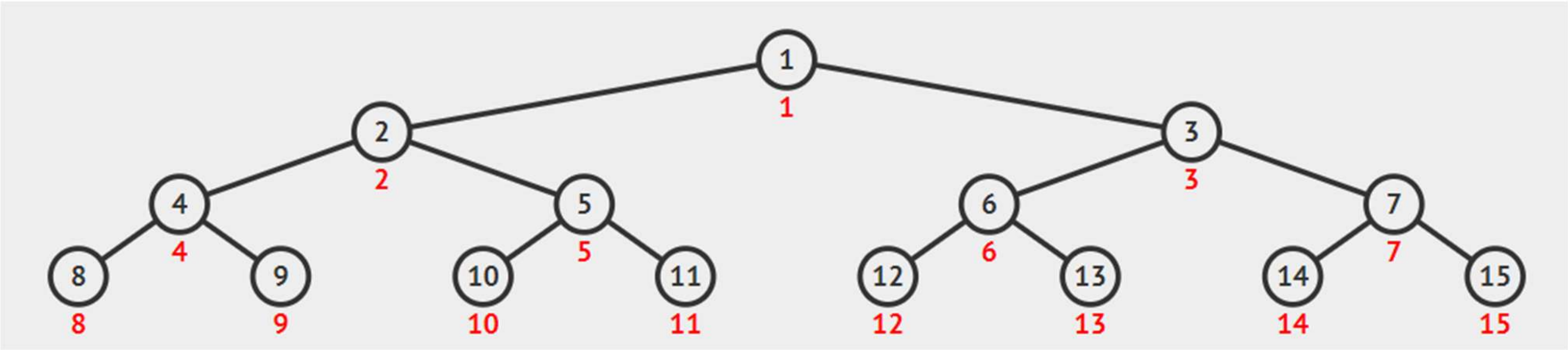
Since N ≤ 20, the BST can only be up to height 19. (20 layers)

The BST would always be a ‘subset’ of a **complete binary tree** of height 19.

C1

Example

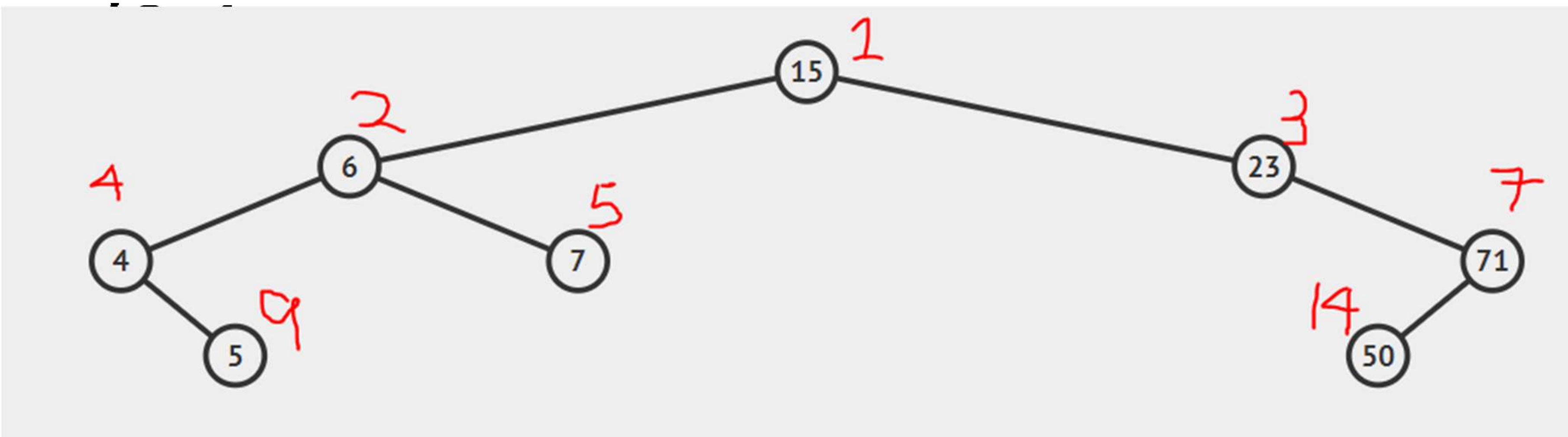
We can label the complete binary tree similar to a heap.

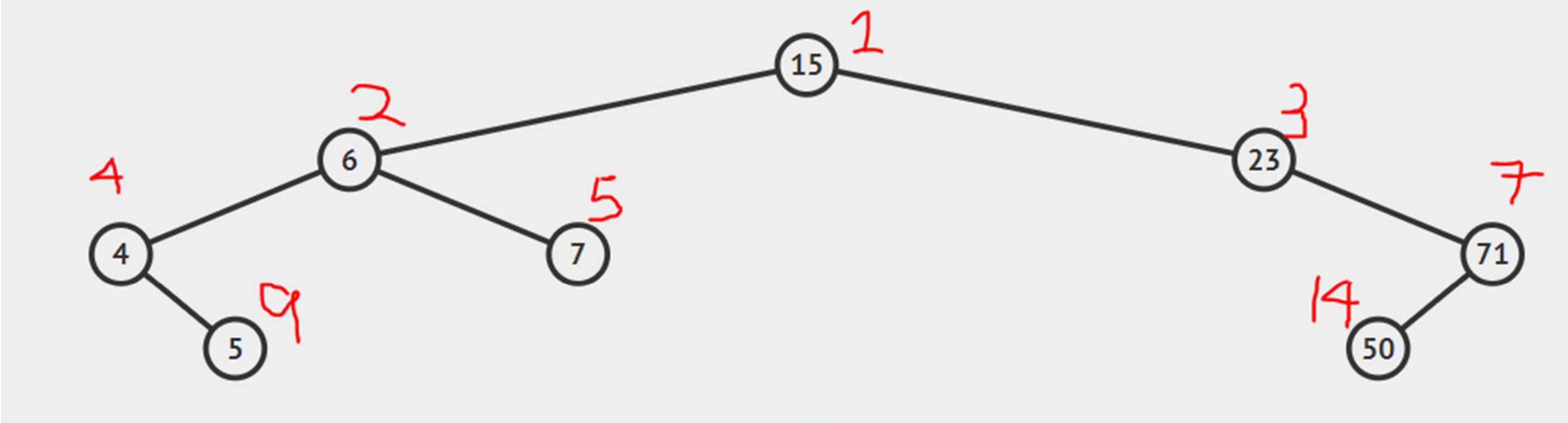


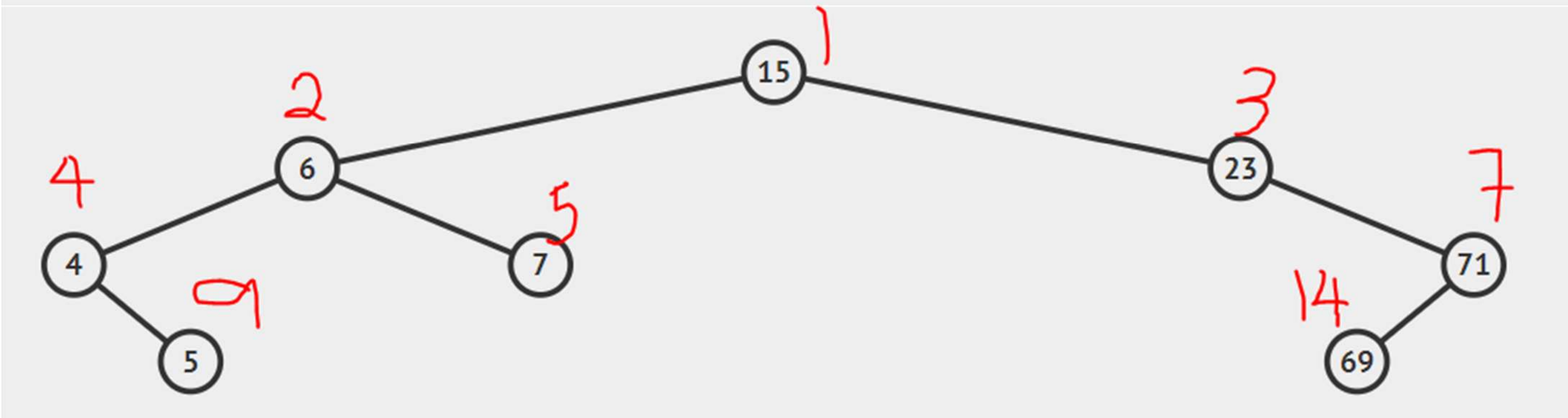
C1

Example

root = **1** left(x) = **x*2** right(x) =




15
1
6
2
23
3
4
4
7
5
71
7
5
9
50
14


15
1
6
2
23
3
4
4
7
5
71
7
5
9
69
14

C1

Approach 2 - Equality check

Two BSTs are equal if the **structure** and the **value** of vertices are equal.

- Structure is the same
  - Check if the two BST share the same ‘labelling’
- Value is the same
  - Check if every vertex has the same value

C1

Implementation

- Generate key value pairs of (label, value) for each BST
- Check of all the pairs are identical for both BST
- Hash Table or Large Array
  - Hash Table is preferred

C1

Implementation

- Generate key value pairs of (label, value) for each BST
- Alternatively, sort all the pairs of the two BST and check if they are identical

C1

Time Complexity

- $O(N^2)$ to generate the BST
- $O(N)$ in subsequent section if you use Hash Table/Large Array of 2^N elements
- $O(N \log N)$ if sort + check

In both cases, $O(N^2)$ total.

C1 Comments

- Hard question
- Fairly few people got the full solution
- Differentiates between A and A+
- Intended to be challenging
- However, most are expected to get **8 marks for N ≤ 3**

From Matching to Connectivity

CS2040C AY1819 Sem 1

C3

You are given a list of **N** rectangles and their coordinates. (**N** ≤ 2000)

Each **rectangle** is a **vertex**.

Two rectangles share an **edge** of **weight 3** *if they overlap*.

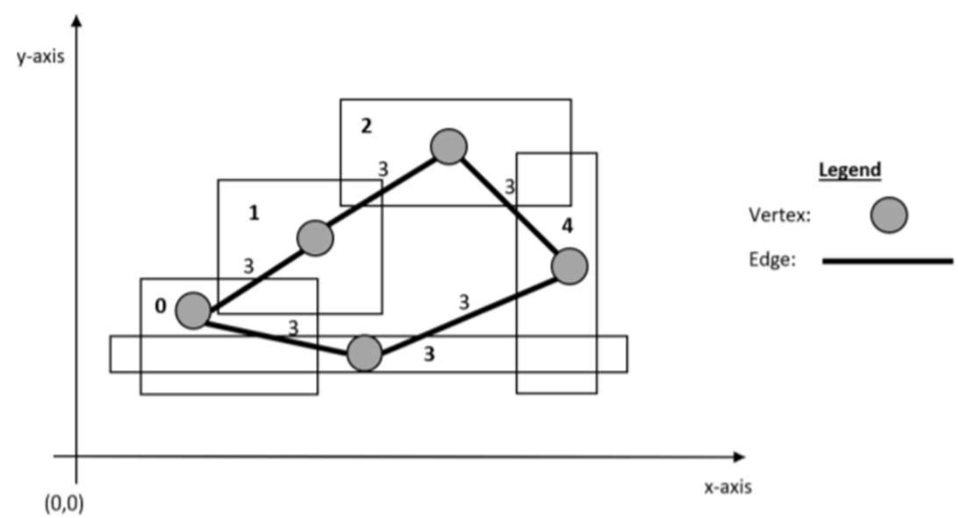


Figure 4: Graph modelled using entities in Figure 3

C3

Given this modelling, you want to find the shortest path from a starting rectangle **S**, to another rectangle **T**.

Eg: **S** = 1, **T** = 4

Cost: 6 (minutes)

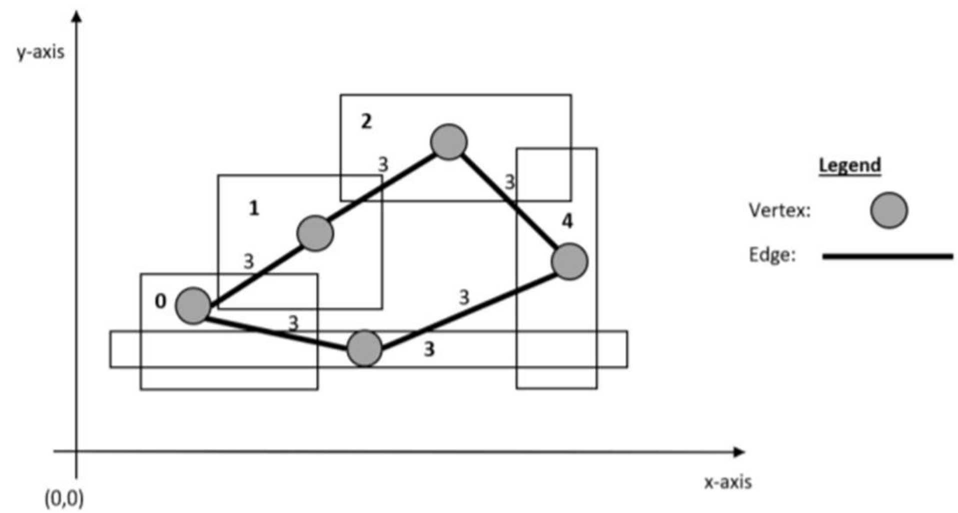


Figure 4: Graph modelled using entities in Figure 3

C3

a) If the input contains N rectangles, how many vertices will there be in the graph formed using this modelling?

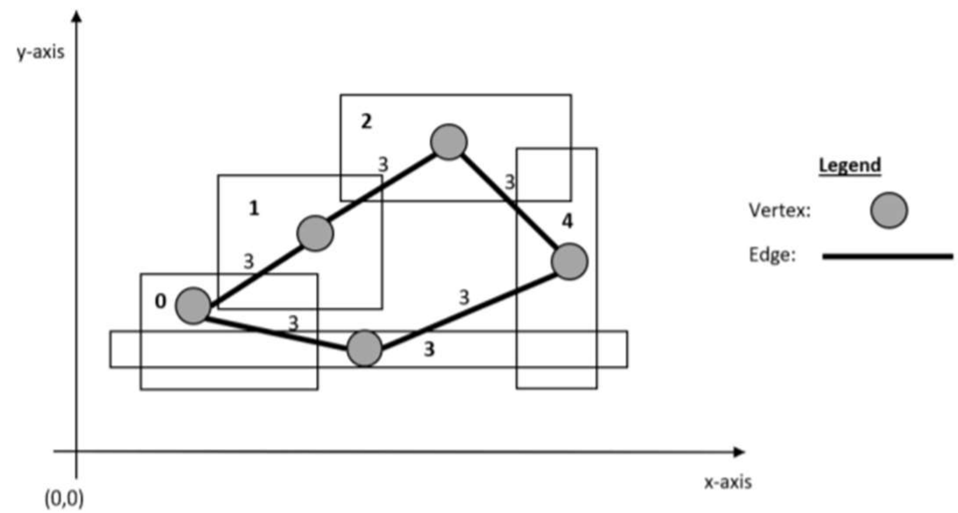


Figure 4: Graph modelled using entities in Figure 3

C3

b) If the input contains **N** rectangles, what is the maximum number of edges in the graph formed using **this modelling**?

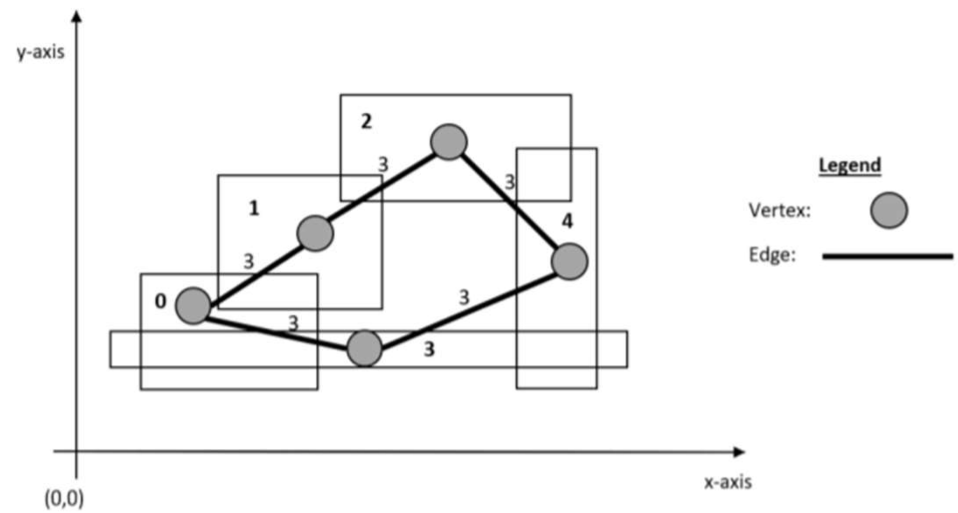


Figure 4: Graph modelled using entities in Figure 3

C3

c) Which of the following would you choose to **store the graph** formed using this modelling: *Edge List*, *Adjacency Matrix* or *Adjacency List*?

Explain your answer.

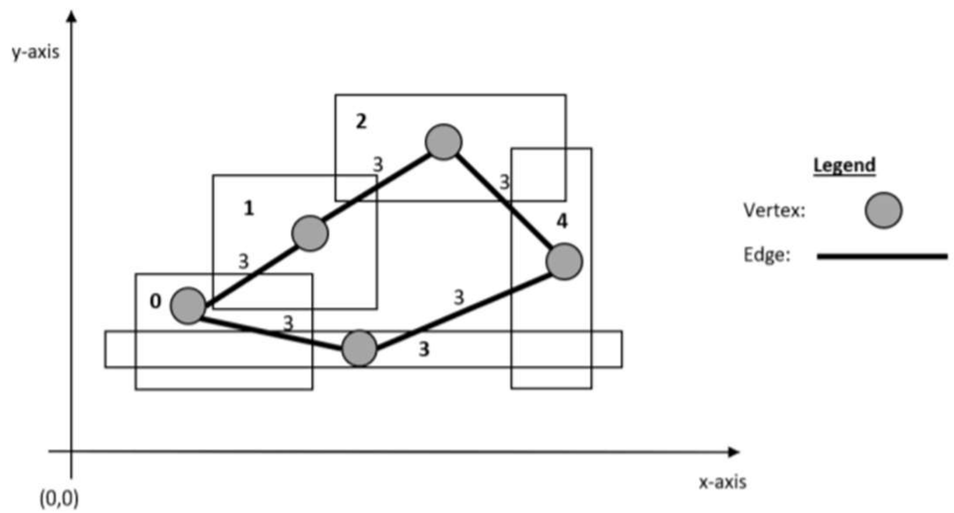


Figure 4: Graph modelled using entities in Figure 3

C3

d) The problem we are trying to solve is similar to a graph problem that has been covered in CS2040C known as … ?

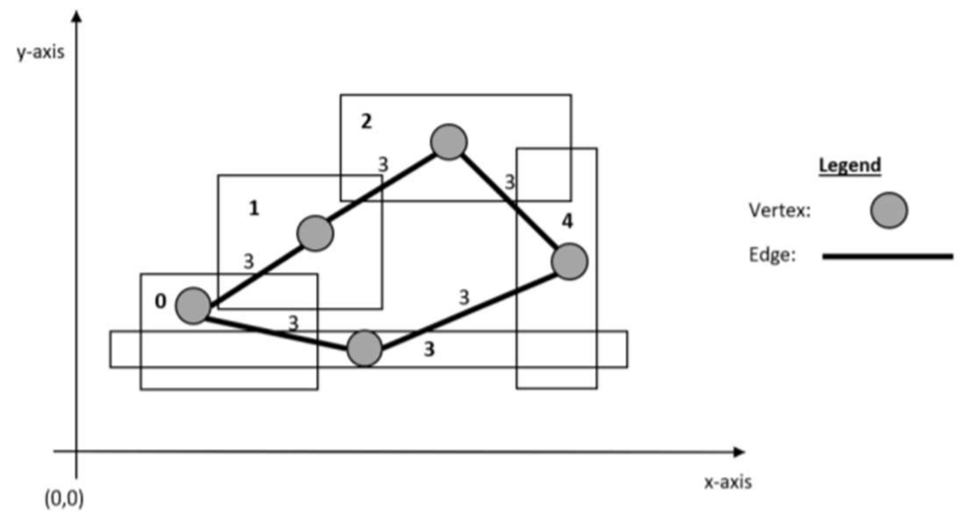


Figure 4: Graph modelled using entities in Figure 3

C3

A function to check whether 2 rectangles overlap is assumed to be provided.

Naive approach

Generate the graph by checking if every pair of rectangles overlap in **$O(N^2)$**.

We will have up to **$O(N^2)$** edges of weight 3.

Perform Dijkstra's algorithm from vertex **S** to **T**.

Time Complexity: **$O(N^2 \log N)$**

C3

Improved approach

Generate the graph by checking if every pair of rectangles overlap in $\mathbf{O(N^2)}$.

Instead, construct up to $\mathbf{O(N^2)}$ ***unweighted*** edges.

Perform **BFS algorithm** from vertex **S** to **T**.

Output the number of edges used **multiplied by 3**.

Time Complexity: $\mathbf{O(N^2)}$

Admin Stuff

Approach to Section C

- Mainly testing on application
  - Thinking
  - Problem solving ability
- C++ syntax/skills is tested, but **not the focus**
- Communicate your algorithm **clearly**, without ambiguity

Approach to Section C

1. Does your algorithm solve the problem?
   - If Yes, what is the time complexity?
   - Assign marks based on time complexity
2. Did the algorithm miss out any edge cases?
   - Deduct marks depending on severity
3. Did you write proper C++ code?
   - Deduct ~25% if all pseudo?

Approach to Section C - Sorting

- Sorting algorithm
  - STL, counting... etc
- What to sort by?
  - Custom comparator is **important**
  - How the sorted result is used.
- Binary search (advised to write pseudo)
  - “Binary search the sorted array A for the **first occurrence** of value **X**”

Approach to Section C - Data Structure

- What data structure**(s)** to use?
  - Time complexity
- What goes inside?
  - Comments will help us understand if you are not confident in your code
  - Eg: This list will store the indexes in the array where the value is positive

Approach to Section C - Graph

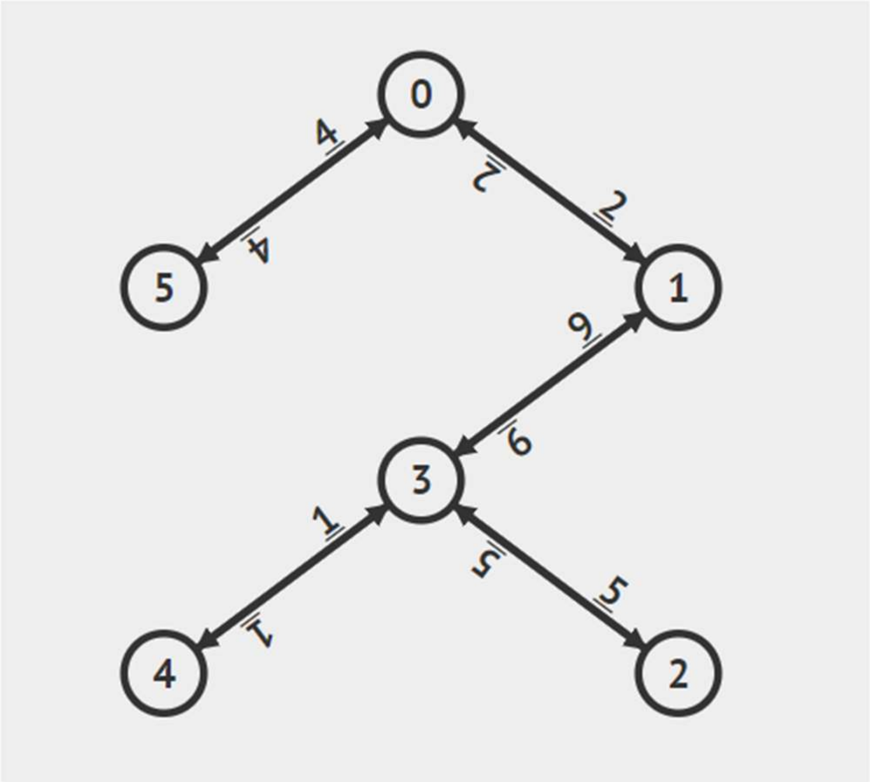
- Construct the Graph
  - What are the vertices?
  - What are the edges?
- Run graph algorithms
  - Denote the starting point for SSSP
- How to use the result of SSSP
- (Maybe) construct multiple graphs?

SSSP Quiz

Q1: Tree

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

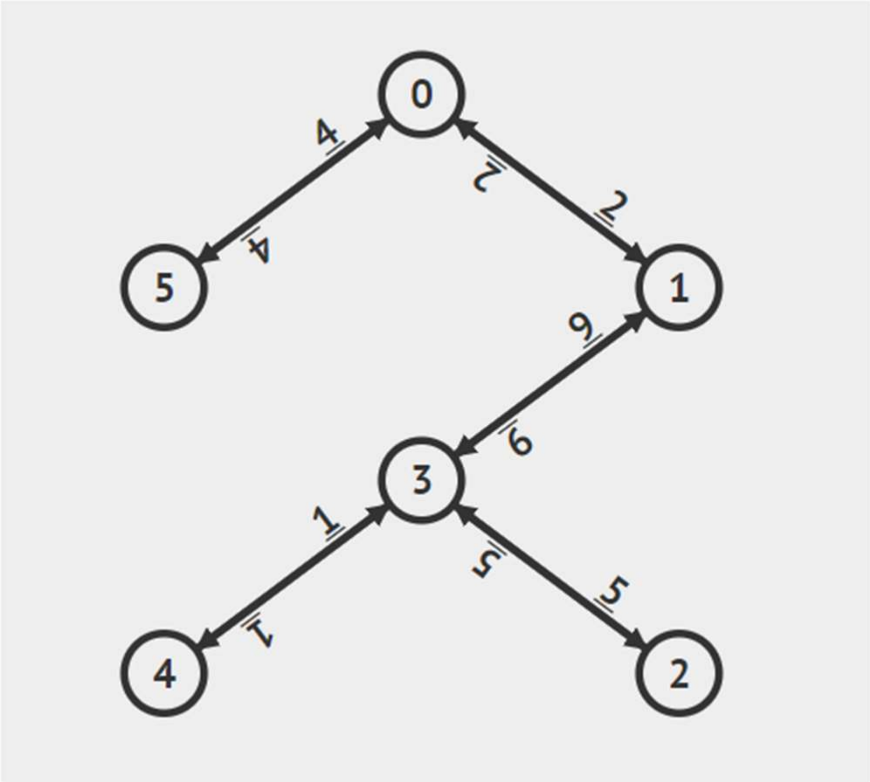
A. DFS
B. BFS
C. Bellman Ford
D. Dijkstra
E. Modified Dijkstra



Q1: Tree

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

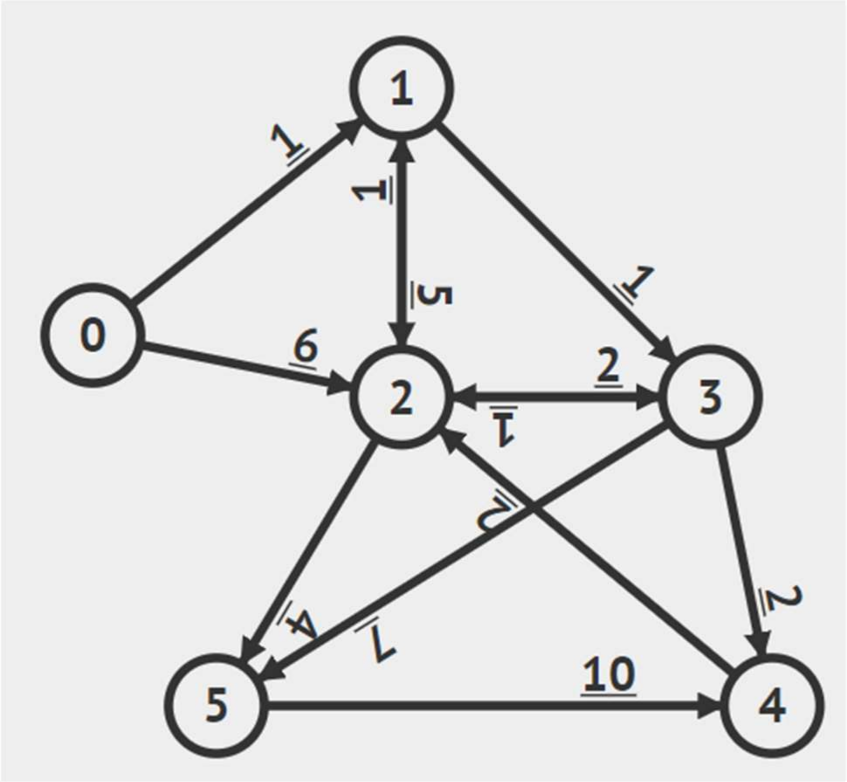
A. DFS (faster)
B. BFS (faster)
C. Bellman Ford
D. Dijkstra
E. Modified Dijkstra



Q2: Small General Graph

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

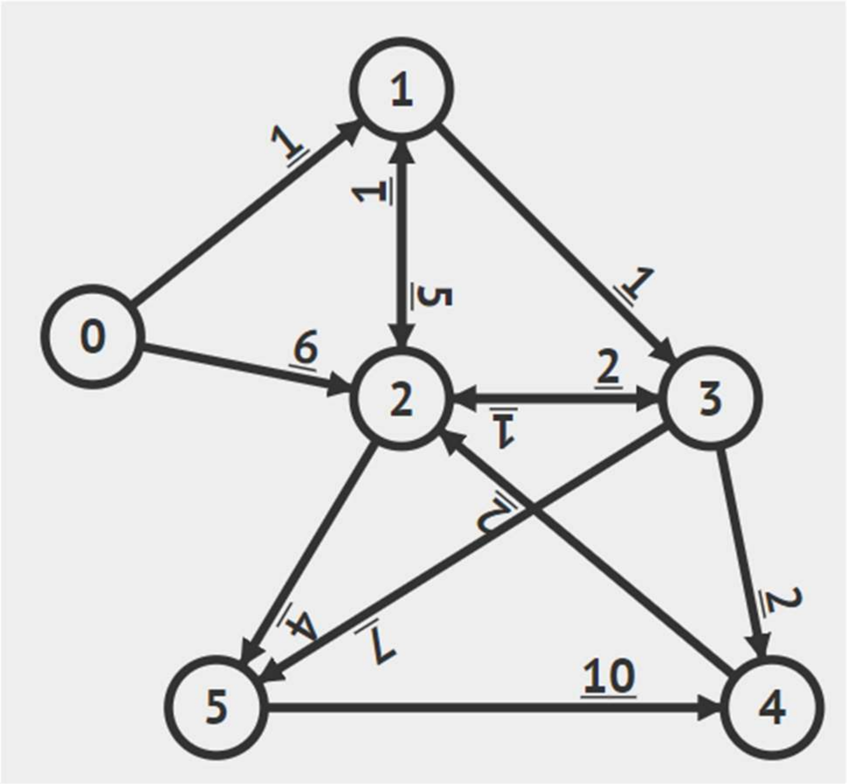
A. DFS
B. BFS
C. Bellman Ford
D. Dijkstra
E. Modified Dijkstra



Q2: Small General Graph

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

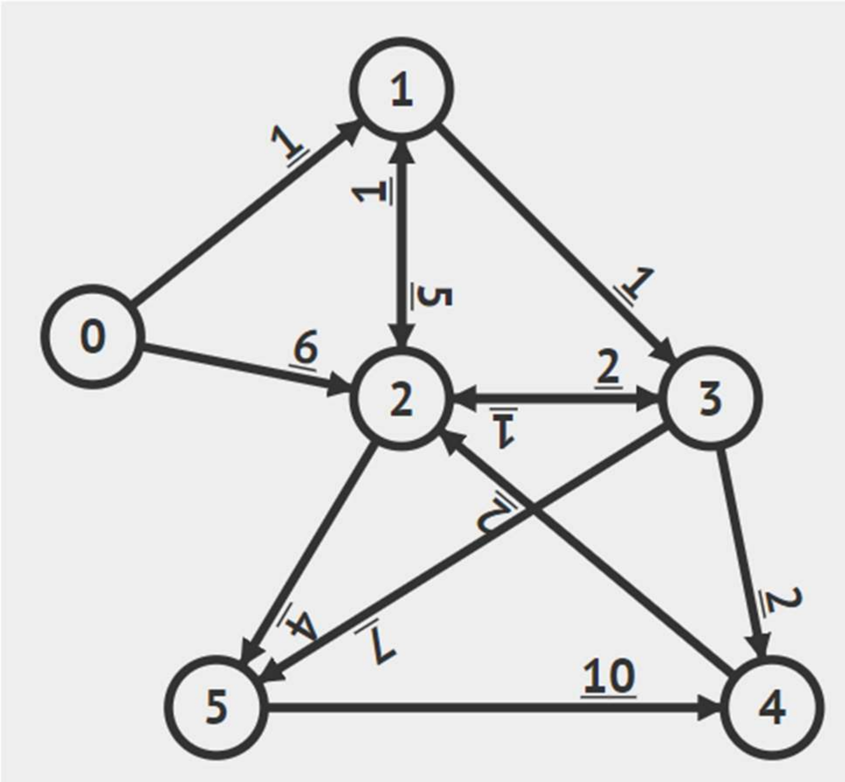
A. DFS
B. BFS
C. Bellman Ford
D. Dijkstra
E. Modified Dijkstra



Q3: General Graph

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

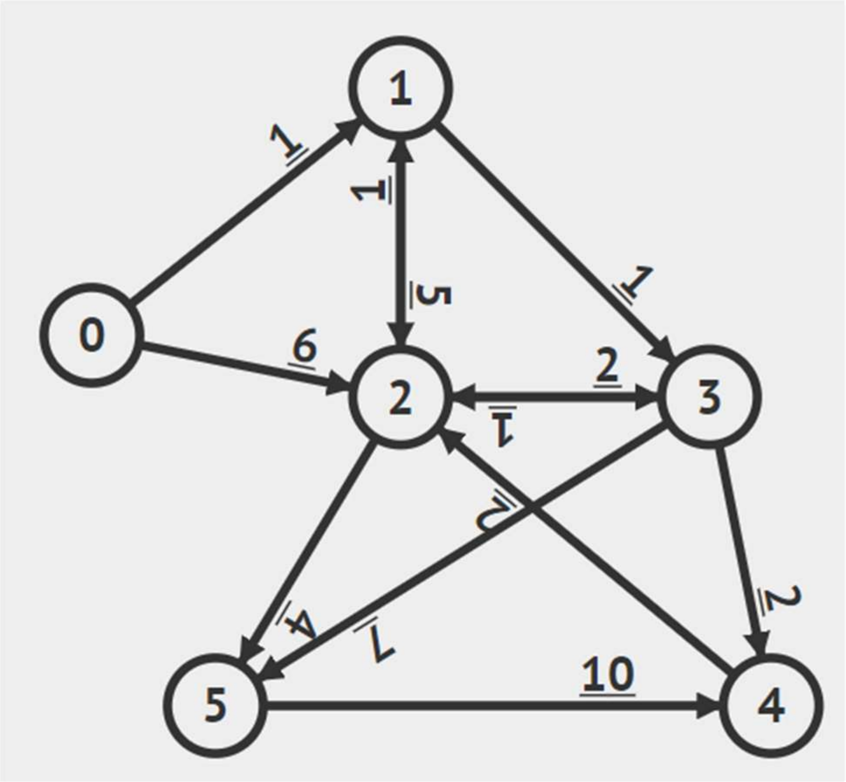
A. DFS
B. BFS
C. Bellman Ford
D. Dijkstra
E. Modified Dijkstra



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?

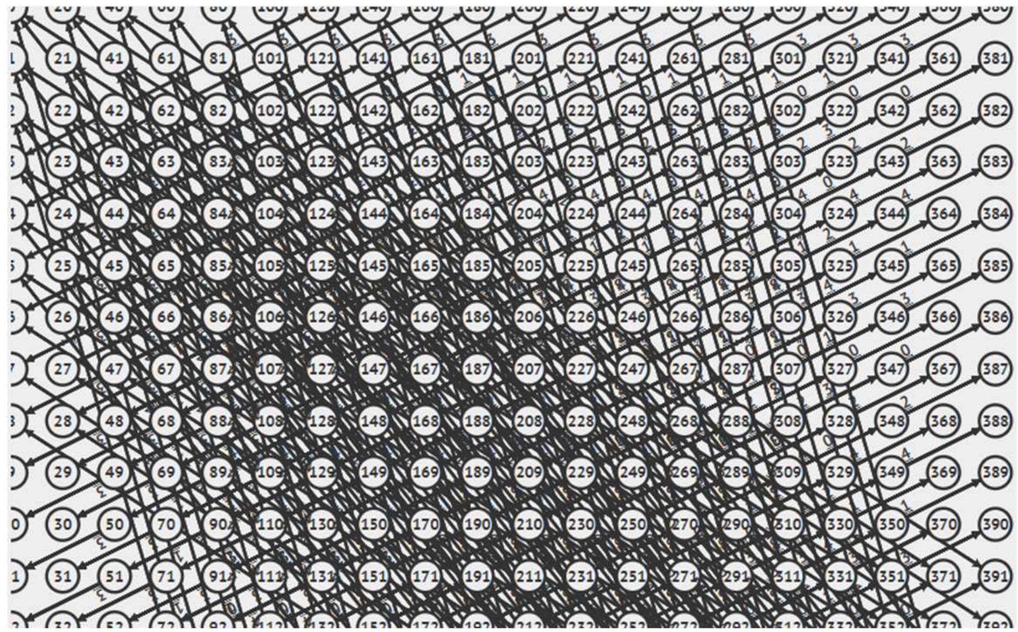
A. DFS
B. BFS
C. Bellman Ford
D. Dijkstra
E. Modified Dijkstra

Some large general graph :O

Q4: Large General Graph

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

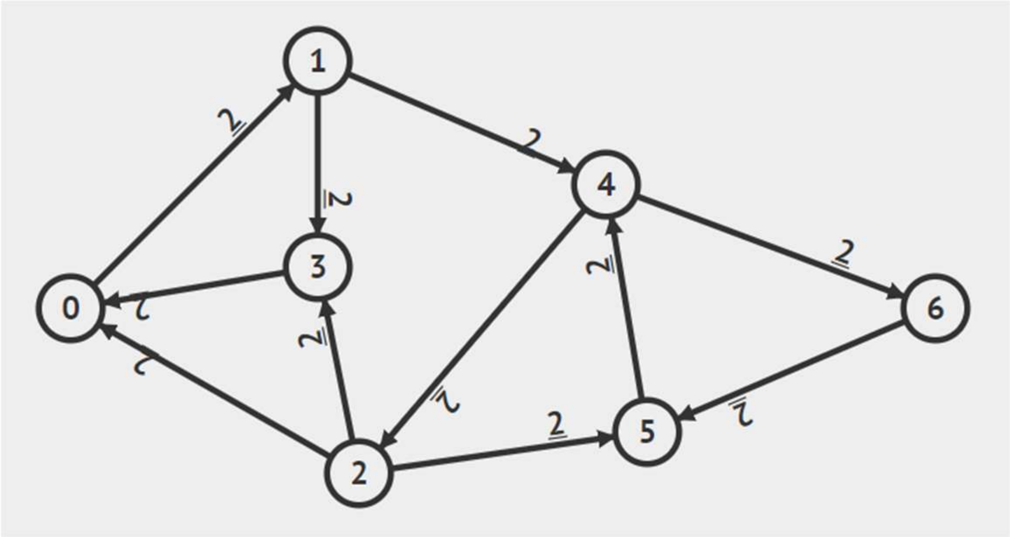
A. DFS
B. BFS
C. Bellman Ford
D. Dijkstra
E. Modified Dijkstra



Q5: Same Edge Weights

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

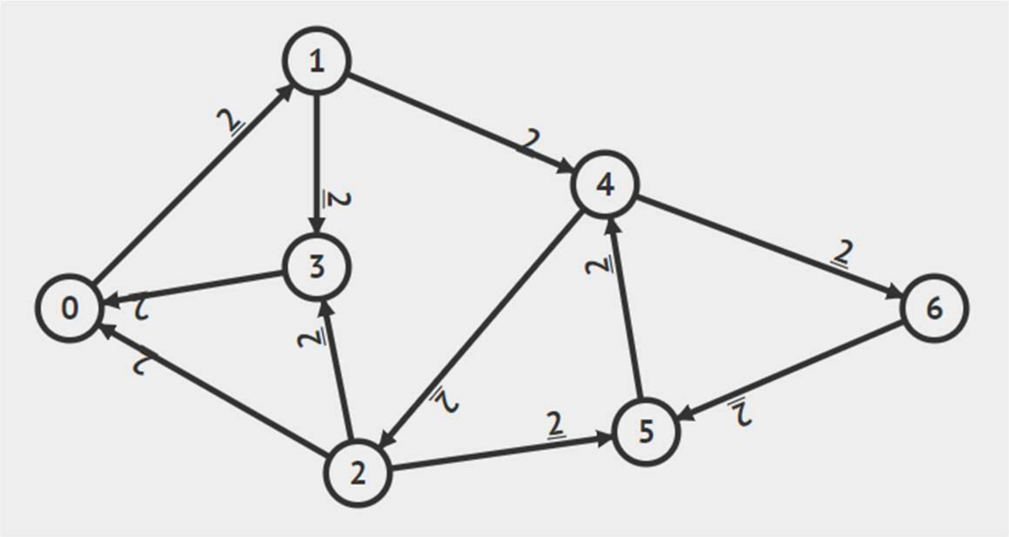
A. DFS
B. BFS
C. Bellman Ford
D. Dijkstra
E. Modified Dijkstra



Q5: Same Edge Weights

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

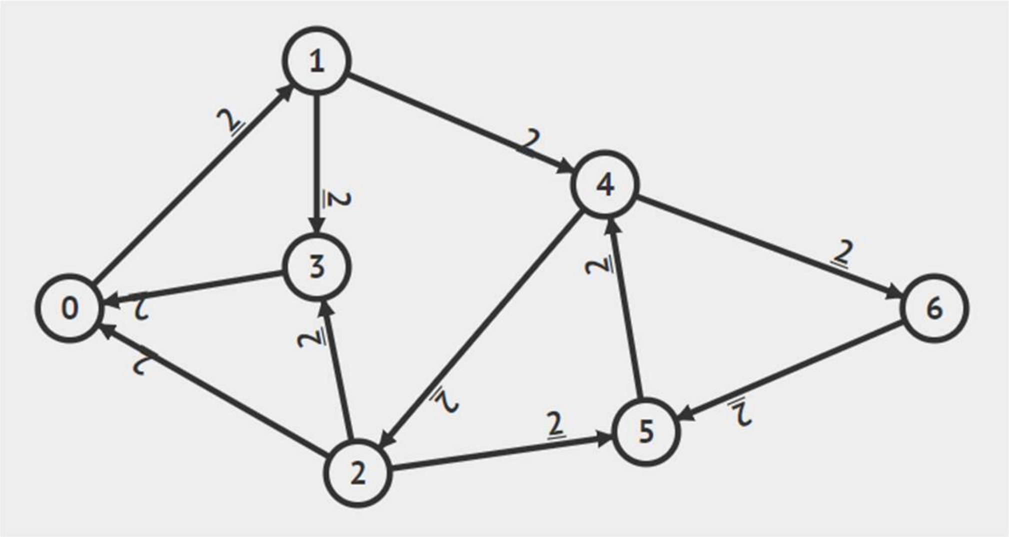
A. DFS
B. BFS
C. Bellman Ford
D. Dijkstra
E. Modified Dijkstra



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.**



Class Photo :)