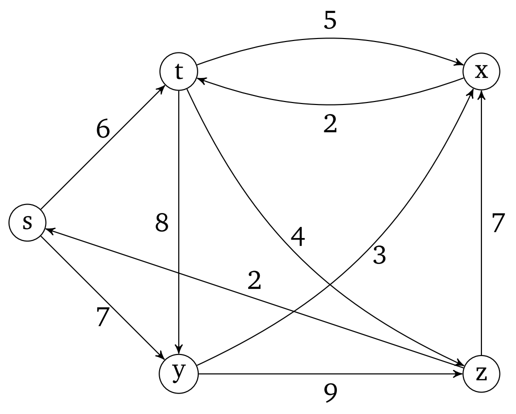
# Section 08: Solutions

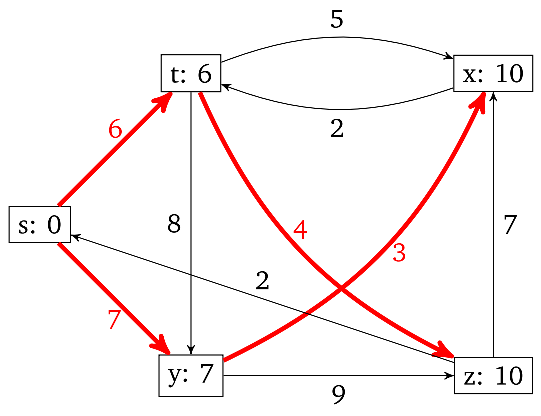
## 1. Simulating Dijkstra's

(a) Consider the following graph:



Suppose we run Dijkstra's algorithm on this graph starting with vertex $s$. What are the final costs of each vertex and the shortest paths from $s$ to each vertex?

**Solution:**

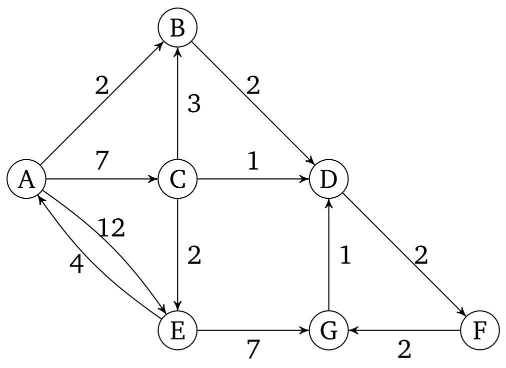


The table below shows the steps of running Dijkstra's algorithm.

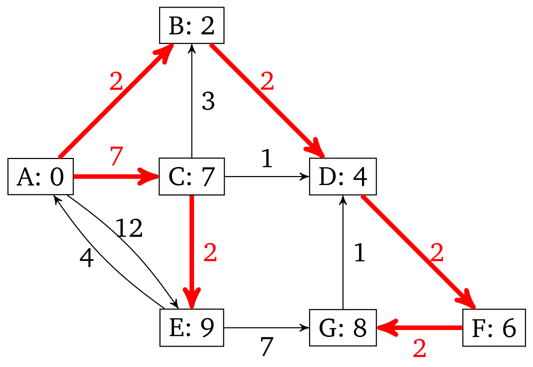
| Vertex | Distance | Predecessor | Processed |
|---|---|---|---|
| s | 0 | – | YES |
| t | 6 | s | YES |
| y | 7 | s | YES |
| z | 10 | t | YES |
| x | 10 | y | YES |

Note that the order of the fourth and fifth vertices are interchangable, since their costs are the same.

(b) Here is another graph. What are the final costs and shortest paths if we run Dijkstra's starting on node $A$?

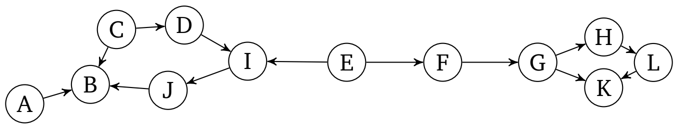


**Solution:**



## 2. Practicing topological sort

Find a topological sort of the following graph:

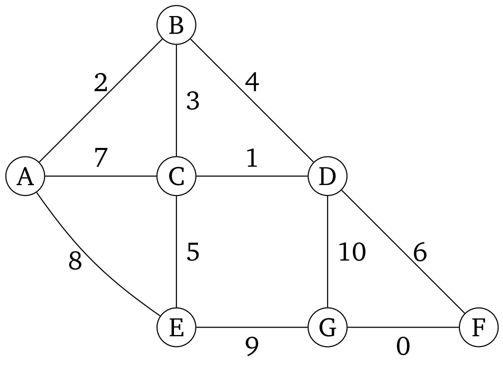


**Solution:**

One possible solution: E, F, G, H, L, K, A, C, D, I, J, B

## 3. Minimum spanning trees

Consider the following graph:



(a) What happens if we run Prim's algorithm starting on node $A$? What are the final costs and edges selected? Give the set of edges in the resulting MST.

**Solution:**

| Step | Components | Edge |
|---|---|---|
| 1 | {A} {B} {C} {D} {E} {F} {G} | (A,B) |
| 2 | {A,B} {C} {D} {E} {F} {G} | (B,C) |
| 3 | {A,B,C} {D} {E} {F} {G} | (C,D) |
| 4 | {A,B,C,D} {E} {F} {G} | (C,E) |
| 5 | {A,B,C,D,E} {F} {G} | (D,F) |
| 6 | {A,B,C,D,E,F} {G} | (F,G) |

(b) What happens if we run Prim's algorithm starting on node $E$? What are the final cost and edges selected? Give the set of edges in the resulting MST.

**Solution:**

| Step | Components | Edge |
|---|---|---|
| 1 | {A} {B} {C} {D} {E} {F} {G} | (E,C) |
| 2 | {C,E} {A} {B} {D} {F} {G} | (C,D) |
| 3 | {C,D,E} {A} {B} {F} {G} | (C,B) |
| 4 | {B,C,D,E} {A} {F} {G} | (B,A) |
| 5 | {A,B,C,D,E} {F} {G} | (D,F) |
| 6 | {A,B,C,D,E,F} {G} | (F,G) |

(c) What happens if we run Prim's algorithm starting on *any* node? What are the final costs and edges selected? Give the set of edges in the resulting MST.

**Solution:**

The answer would be the same as the one we get above, since for each node, we always choose the smallest-weight edge that links to it.

(d) What happens if we run Kruskal's algorithm? Give the set of edges in the resulting MST.

**Solution:**

We'll use this table to keep track of components and edges we processed. The edges are listed in an order sorted by weight.

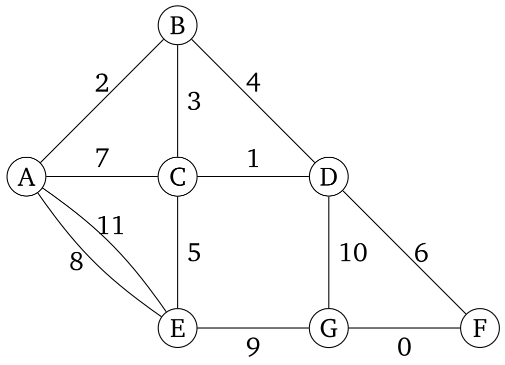
| Step | Components | Edge | Include? |
|---|---|---|---|
| 1 | | (F,G) | |
| 2 | | (C,D) | |
| 3 | | (A,B) | |
| 4 | | (B,C) | |
| 5 | | (B,D) | |
| 6 | | (C,E) | |
| 7 | | (D,F) | |
| 8 | | (A,C) | |
| 9 | | (A,E) | |
| 10 | | (E,G) | |
| 11 | | (D,G) | |

After executing Kruskal's algorithm on the above graph, we get

| Step | Components | Edge | Include? |
|---|---|---|---|
| 1 | {A} {B} {C} {D} {E} {F} {G} | (F,G) | Yes |
| 2 | {A} {B} {C} {D} {E} {F,G} | (C,D) | Yes |
| 3 | {A} {B} {C,D} {E} {F,G} | (A,B) | Yes |
| 4 | {A,B} {C,D} {E} {F,G} | (B,C) | Yes |
| 5 | {A,B,C,D} {E} {F,G} | (B,D) | No |
| 6 | {A,B,C,D} {E} {F,G} | (C,E) | Yes |
| 7 | {A,B,C,D,E} {F,G} | (D,F) | Yes |
| 8 | {A,B,C,D,E,F,G} | (A,C) | No |
| 9 | {A,B,C,D,E,F,G} | (A,E) | No |
| 10 | {A,B,C,D,E,F,G} | (E,G) | No |
| 11 | {A,B,C,D,E,F,G} | (D,G) | No |

The resulting MST is a set of all edges marked as *Include* in the above table.

(e) Suppose we modify the graph above and add a heavier parallel edge between A and E, which would result in the graph shown below. Would your answers for above subparts (a, b, c, and d) be the same for this following graph as well?

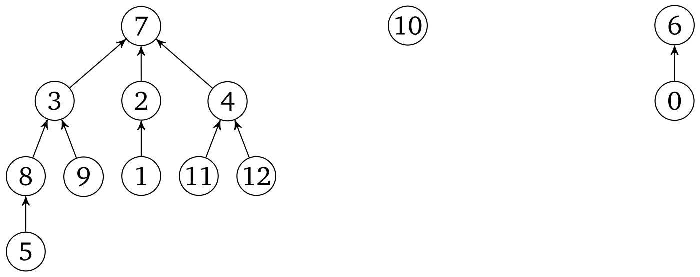


**Solution:**

The steps are exactly the same, since we don't consider the heavier edge when there are parallel edges. The reason is that the heavier edge would be considered as the best edge when there is a lighter one that can be added to the graph.

## 4. Disjoint sets

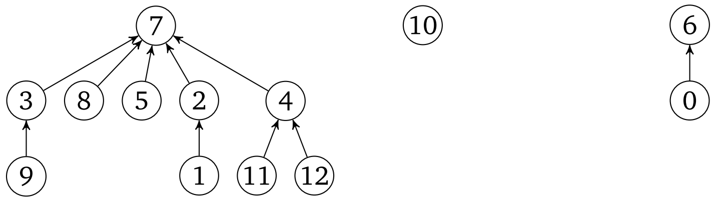
Consider the following trees, which are a part of a disjoint set data-structure:



For these problems, use both the **union-by-rank** and **path compression** optimizations. Assume that the first tree has rank $3$, the second has rank $0$ and the last has rank $1$.

(a) Draw the resulting tree(s) after calling `findSet(5)` on the above. What value does the method return?

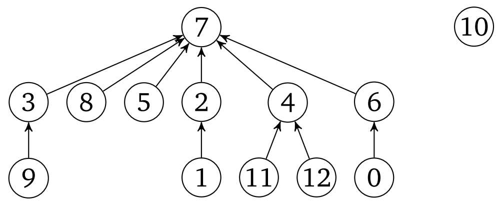
**Solution:**



The method returns 7, which is the root of the tree.

(b) Draw the final result of calling `union(2,6)` on the result of part a.

**Solution:**



We merge the second tree to the first since the rank of the first tree is 2, while that of the second tree is 1.

## 5. Design Problem: Pathfinding in mazes

Suppose we are trying to design a maze within a 2d top-down video-game. The world is represented as a grid, where each tile is either an impassable wall, an open space a player can pass through, or a *wormhole*. On each turn, the player may move one space on the grid to any adjacent open tile. If the player is standing on a wormhole, they can instead use their turn to teleport themselves to the other end of the wormhole, which is located somewhere else on the map.

Now, suppose the there are several coins scattered throughout the map. Your goal is to design an algorithm that finds a path between the player and some coin in the fewest number of turns possible.

Describe how you would represent this scenario as a graph (what are the vertices and edges? Is this a weighted or unwighted graph? Directed or undirected?). Then, describe how you would implement an algorithm to complete this task.

**Solution:**

We can represent this as an undirected, unweighted graph where each tile is a vertex. Edges connect tiles we can travel between. When we have a wormhole, we add an extra edge connecting that wormhole tile to the corresponding end of the wormhole.

Because it takes only one turn to travel to each adjacent tile, there is actually no need to store edge weights: it costs an equal amount to move to the next vertex.

All paths are bidirectional, so we can also use an undirected graph. (If there are paths or wormholes that are one-way, we can switch to using a directed graph).

To find the shortest path, we can run BFS starting with the player and stop the moment we hit a coin.

(We can use other algorithms like DFS or Dijkstra's algorithm if we're careful, but those would be less efficient.)

## 6. Design Problem: Negative Edge Weights

If you enjoy reading Pokémon jokes, you can read the flavor text to understand where the graph problem comes from. Otherwise, you can skip those parts and just read the formal statements.

**Flavor Text:** You and your trusty Pikachu have made it halfway through Viridian Forest, but things have taken a turn for the worst. That last Weedle poisoned your Pikachu, and you're all out of antidotes.

In the Pokémon world, the poison doesn't do any damage as long as you stay *perfectly still*. But every time you take a step, the poison does a little bit of damage to your poor friend Pikachu.

Thanks to Bulbapedia[1], you know the exact map of Viridian Forest. Knowing that each step will cost your Pikachu exactly one of its precious hit points, you will need to find an efficient path through the forest. [2]

**Formal Statement:** In a video game you are playing, each step you take costs a character (Pikachu) one unit of health. You have a map of the level (Viridian Forest) – your goal is to reach the end of the level (marked on your map) while losing as little health as possible.

(a) Describe a graph and an algorithm run on that graph to find the path through the forest to save as many of Pikachu's hit points as possible (i.e. the path with the fewest number of steps). **Solution:**

Have a vertex for each possible location in Viridian Forest, and an edge between every two vertices we can move between in one step. Since Pikachu loses the same amount of hit points per step, we can just leave the graph unweighted.

Since the graph is unweighted, we can just run BFS, starting from our current location, with a target of the end of Viridian Forest.

You could use either a directed graph or an undirected graph for this part.

(b) **Flavor Text:** You run your algorithm and come to a devastating realization – the edge of Viridian Forest is at least 25 steps away, and Pikachu has only 20 hit points left. If you just walk to the end of the forest, Pikachu will faint before reaching the next Pokémon Center. So you come up with a backup plan. Returning to Bulbapedia, you see there is a potion just a little bit out of the way of the fastest path.

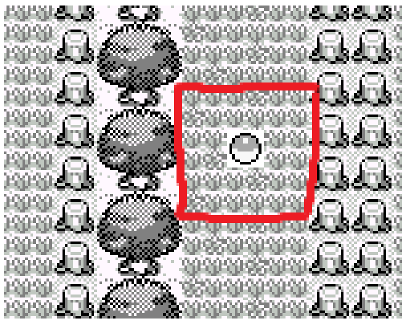
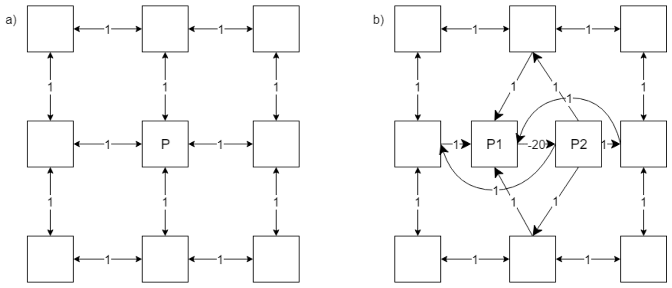
Brock tells you he knows how to update your graph to find the best path now. He says he'll add a dummy vertex to the graph where the potion is and connect up the new vertex with a (directed) edge of length $-20$, to represent undoing the loss of 20 hit points.

**Formal Statement:** You realize your character doesn't have enough health to make it to the edge of the forest. But you know there is a healing item (a "potion") somewhere in the forest, that will give you back $20$ units of health.

A friend (Brock) suggests the following update: add a dummy vertex to the graph where the healing item is and connect up the new vertex with a (directed) edge of length $-20$, to represent undoing the loss of 20 hit points.

[1]Like Wikipedia, but for Pokémon!

[2]Don't worry about running into wild Pokémon. For some reason you have a huge number of repels. Next time, maybe invest in full heals or potions instead.

9 spots in Viridian Forest, the corresponding vertices before Brock's transformation and the same vertices after the transformation.

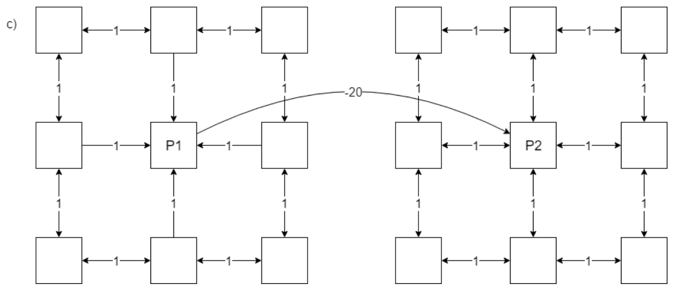
Tell Brock why his representation isn't quite going to work (hint: you can only use the potion once. What happens if the potion edge is part of a cycle?).

**Solution:**

The potion edge is part of a cycle.

What happens if you go around the cycle repeatedly? Each time the distance you've gone gets "shorter!" So no matter how many times you've gone around the cycle you should go around once more, and you'll be able to find an even shorter path from your current location to the edge of the forest. With Brock's representation, the shortest path isn't even defined!

(c) You convince Brock to change the graph representation. You'll now have two copies of the original Viridian Forest graph, in copy 1 the potion is still unused. In copy 2, the potion is no longer there. You add an edge of weight $-20$ from copy 1 to copy 2 at the location of the potion (crossing that edge represents using that potion). His new graph looks something like this.



Brock says he'll start running Dijkstra's. Should you trust the output?

**Solution:**

No, Dijkstra's isn't guaranteed to work when there are negative edges. Poor Brock. He knows so much about rock Pokemon but so little about algorithms.

Luckily your Pokédex gets good data service in Viridian Forest, and you look up the Bellman-Ford algorithm for finding shortest paths with negative edge weights and find the new best path.

(d) **Challenge Problem**: Misty says she knows about another potion over there somewhere. Describe how to modify the graph to handle both of the potions.

**Solution:**

We now want 4 copies of the graph. One for each of (no potions used, only potion 1 used, only potion 2 used, both potions used). Make edges of weight $-20$ to connect these in the same way as you did in the last part.

To make it easier to choose a final destination, add a dummy destination vertex. Then add a weight $0$ edge from each copy of the edge of the forest to the dummy destination.