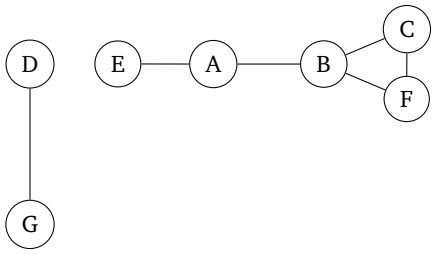
Section 06: Solutions

Section Problems

1. Graph properties

(a) Consider the *undirected, unweighted* graph below.



Answer the following questions about this graph:

(i) Find V, E, $|V|$, and $|E|$.

(ii) What is the maximum *degree* of the graph?

(iii) Are there any cycles? If so, where?

(iv) What is the maximum length simple path in this graph?

(v) What is one edge you could add to the graph that would increase the length of the maximum length simple path of the new graph to 6?

(vi) What are the *connected components* of the graph?

Solution:

(i) $V = \{A, B, C, D, E, F, G\}$ and $E = \{(D, G), (E, A), (A, B), (B, F), (F, C), (C, B)\}$. This means that $|V| = 7$ and $|E| = 6$.

(ii) The vertex with the max degree is B, which has a degree of 3.

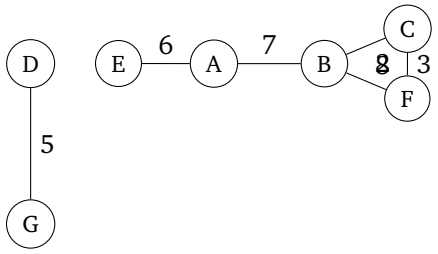
(iii) There is indeed a cycle, between B, C, and F.

(iv) There are multiple simple paths with the maximum length: $(E, A), (A, B), (B, F), (F, C)$ or $(E, A), (A, B), (B, C), (C, F)$ or their inverses (starting at C or F instead of E).

(v) We could add the edge (D, E).

(vi) One connected component is $\{D, G\}$. Another one is $\{E, A, B, C, F\}$.

(b) Consider the *undirected, weighted* graph below.



Answer the following questions about this graph:

(i) What is the path involving the least number of nodes from E to C? What is its cost?

(ii) What is the minimum cost path from E to C? What is its cost?

(iii) What is the minimum length path from E to C? What is its length?

Solution:

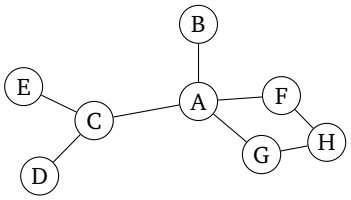
(i) The path with the least number of nodes is $(E, A), (A, B), (B, C)$. The cost is 21.

(ii) The minimum cost path is actually $(E, A), (A, B), (B, F), (F, C)$. The cost is 18.

(iii) The path with the shortest length is $(E, A), (A, B), (B, C)$. The length is 3.

2. Graph traversal

(a) Consider the following graph. Suppose we want to traverse it, starting at node A.



If we traverse this using *breadth-first search*, what are *two* possible orderings of the nodes we visit? What if we use *depth-first search*?

For the first ordering, you **must** run through adding/removing things from the queue/stack. To provide the second ordering for each algorithm, you may simply look at the graph.

Solution:

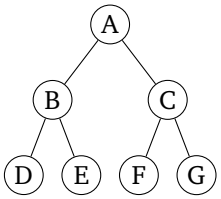
Here are two possible orderings for BFS:

- A, G, F, B, C, H, D, E
- A, C, B, F, G, D, H, E

Here are two possible orderings for DFS:

- A, G, H, F, C, D, E, B
- A, B, C, E, D, F, H, G

(b) Same question, but on this graph:



Solution:

Here are two possible orderings for BFS:

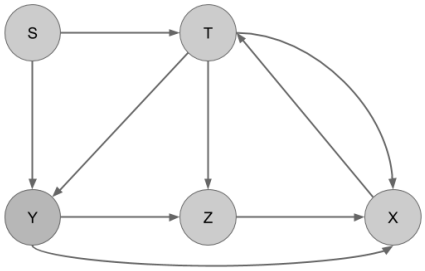
- A, B, C, D, E, F, G
- A, C, B, F, G, D, E

Here are two possible orderings for DFS:

- A, B, D, E, C, F, G
- A, C, G, F, B, E, D

3. Simulating BFS

Consider the following graph:



Run the BFS algorithm to find the shortest paths in this graph starting from vertex s. Draw out the queue of nodes, and also use the table below to keep track of each step in the algorithm. Finally, draw the resulting SPT (shortest path tree) after the algorithm has terminated.

Note: If two nodes enter the queue at the same time, break ties so that the node that comes first alphabetically exits the queue first.

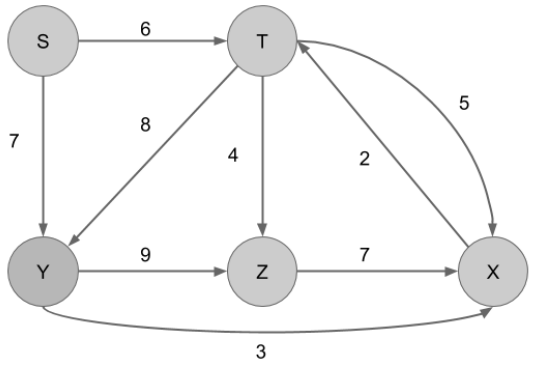
Vertex	Predecessor	Processed
s		
t		
x		
y		
z		

Solution:

Please see the Section slides for a set-by-step walk-through!

4. Simulating Dijkstra's

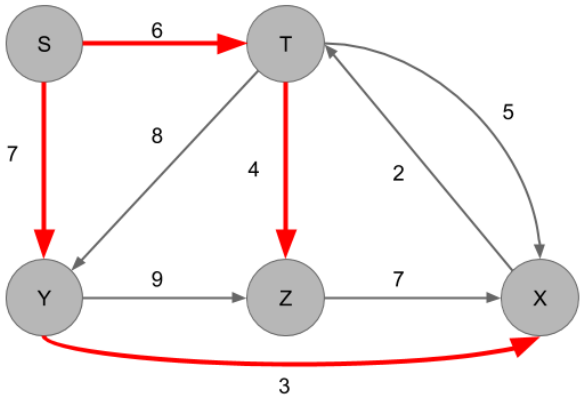
(a) Consider the following graph:



Run Dijkstra's algorithm on this graph starting from vertex s. Use the table below to keep track of each step in the algorithm. Also draw the resulting SPT (shortest path tree) after the algorithm has terminated.

Vertex	**Distance**	**Predecessor**	**Processed**
s			
t			
x			
y			
z			

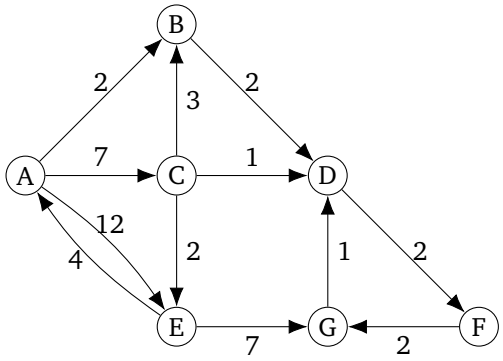
Solution:



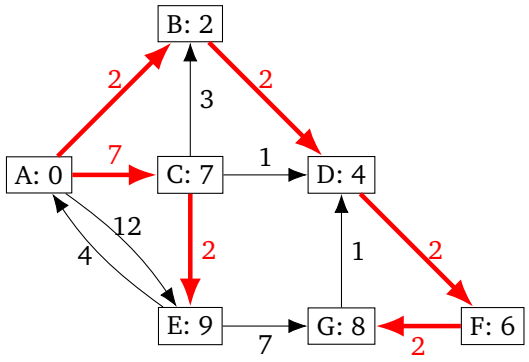
The table below shows the final result of running Dijkstra's algorithm.

Vertex	Distance	Predecessor
s	0	–
t	6	s
x	10	y
y	7	s
z	10	t

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



Solution:



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: DJ Kistra

You've just landed your first big disk jockeying job as "DJ Kistra."

During your show you're playing "Shake It Off," and decide you want to slow things down with "Wildest Dreams." But you know that if you play two songs whose tempos differ by more than 10 beats per minute or if you play only a portion of a song, that the crowd will be very disappointed. Instead you'll need to find a list of songs to play to gradually get you to "Wildest Dreams." Your goal is to transition to "Wildest Dreams" with a playlist of progressively slower songs as quickly as possible (in terms of seconds).

You have a list of all the songs you can play, their speeds in beats per minute, and the length of the songs in seconds.

(a) Describe a graph you could construct to help you solve the problem. At the very least you'll want to mention what the vertices and edges are, and whether the edges are weighted or unweighted and directed or undirected.

Solution:

Have a vertex for each song. Draw a directed edge from song A to song B if (and only if) song B is slower than A, but the difference between their speeds is at most 10 beats per minute. Add a weight equal to the length of song B to each such edge.

Note: You can also argue that the weight should be the length of song A (if you assumed that we haven't started playing "Shake It Off" yet at the beginning of the problem).

(b) Describe an algorithm to construct your graph from the previous part. You may assume your songs are stored in whatever data structure makes this part easiest. Assume you have access to a method `makeEdge(v1, v2, w)` which creates an edge from `v1` to `v2` of weight `w`.

Solution:

```
foreach(Song s1) {
    foreach(Song s2) {
        if(s2.bpm < s1.bpm && |s1.bpm - s2.bpm| <= 10)
        {
            makeEdge(s1, s2, s2.songLength);
        }
    }
}
```

As long as our data structure has an efficient iterator this algorithm will run in $O(|S|^2)$ time since we have a double loop.

(c) Describe an algorithm you could run on the graph you just constructed to find the list of songs you can play to get to "Wildest Dreams" the fastest without disappointing the crowd.

Solution:

Run Dijkstra's from "Shake It Off." When the algorithm finishes, use back pointers from "Wildest Dreams" (and reverse the order) to find the songs to play.

(d) What is the running time of your plan to find the list of songs? You should include the time it would take to construct your graph and to find the list of songs. Give a simplified big-O running time in terms of whatever variables you need.

Solution:

The answer will depend on what you chose in the previous parts. If you used an efficient iterator with a double loop to build your graph, that approach gives a running time of $O(S^2 + E \log S)$.

7. Buggy Dijkstra's

Your best friend tried their hand at writing a more complete Dijkstra's pseudocode. However, you noticed something isn't quite right. Find the bug(s), explain why the behavior isn't correct, and suggest a fix.

```
dijkstraShortestPath(G graph, V start, V end)
    Set known; Map edgeTo, distTo;
    initialize distTo with all nodes mapped to infinity, except start to 0

    while (there are unknown vertices):
        let u be the closest unknown vertex
        known.add(u)
        for each edge (u,v) to unknown v with weight w:
            if (v == end)
                return path // We found the goal node, we're done!
            oldDist = distTo.get(v)
            newDist = distTo.get(u) + w
            if (newDist < oldDist):
                distTo.put(v, w)
                edgeTo.put(v, u)
```

Solution:

There are two bugs. (1) The student is returning as soon as they find the end node. This is incorrect because we haven't processed the end node yet, so we don't know if we really found the shortest path! The fix would be to only exit until after we've marked end as processed. (2) When we find a new shortest path, we are putting in w. This is incorrect because the shortest path must include the weight of the entire path up to v. To fix this, we should be adding newDist, which includes the distTo.get(u).

8. 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.

[1] Like Wikipedia, but for Pokémon!

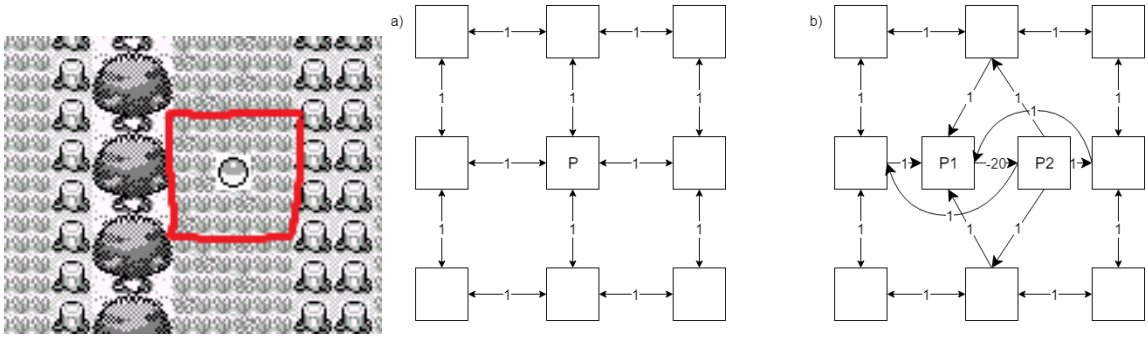
[2] Don't worry about running into wild Pokémon. For some reason you have a huge number of repels. Next time, maybe invest in full heals or potions instead.

(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.



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

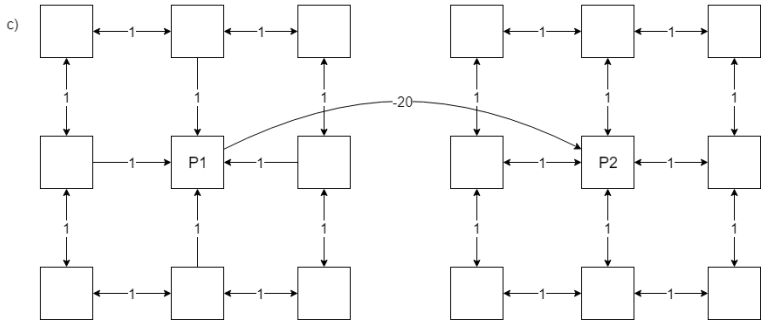
Tell Brock why his representation isn't quite going to work (hint: you can only use the potion once. What happens if the potion edge is part of a cycle?).

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 a second potion somewhere else in the forest. 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.