Shortest Path in Digraphs

Arash Rafiey

October 4, 2016

Computing Shortest Path in Digraphs

Input : We are given a digraph D, where each arc has a weight (non-negative number).

Goal : Find a shortest (cheapest value) path from node s to every other node in D.

Computing Shortest Path in Digraphs

Input : We are given a digraph D, where each arc has a weight (non-negative number).

Goal : Find a shortest (cheapest value) path from node s to every other node in D.

1. Denote the shortest distance from s to u by $d(u)$, and $P(u)$ denotes the last node before u on the shortest path from s to u.

Computing Shortest Path in Digraphs

Input : We are given a digraph D, where each arc has a weight (non-negative number).

Goal : Find a shortest (cheapest value) path from node s to every other node in D.

1. Denote the shortest distance from s to u by $d(u)$, and $P(u)$ denotes the last node before u on the shortest path from s to u.

2. The algorithm maintains a set S of nodes u for which $d(u)$ and $P(u)$ computed correctly.

Computing Shortest Path in Digraphs

Input : We are given a digraph D, where each arc has a weight (non-negative number).

Goal : Find a shortest (cheapest value) path from node s to every other node in D.

1. Denote the shortest distance from s to u by $d(u)$, and $P(u)$ denotes the last node before u on the shortest path from s to u.

2. The algorithm maintains a set S of nodes u for which $d(u)$ and $P(u)$ computed correctly.

3. Initially $S = \{s\}$ and $d(s) = 0$, $P(u) = s$.

Computing Shortest Path in Digraphs

Input : We are given a digraph D, where each arc has a weight (non-negative number).

Goal : Find a shortest (cheapest value) path from node s to every other node in D.

1. Denote the shortest distance from s to u by $d(u)$, and $P(u)$ denotes the last node before u on the shortest path from s to u.

2. The algorithm maintains a set S of nodes u for which $d(u)$ and $P(u)$ computed correctly.

3. Initially $S = \{s\}$ and $d(s) = 0$, $P(u) = s$.

4. For each node $v \in V - S$ find a shortest path P from s to v where all the nodes in $P - v$ are in S (only the last node of P is outside S; v is connected to S by some edge uv).

Computing Shortest Path in Digraphs

Input : We are given a digraph D, where each arc has a weight (non-negative number).

Goal : Find a shortest (cheapest value) path from node s to every other node in D.

1. Denote the shortest distance from s to u by $d(u)$, and $P(u)$ denotes the last node before u on the shortest path from s to u.

2. The algorithm maintains a set S of nodes u for which $d(u)$ and $P(u)$ computed correctly.

3. Initially $S = \{s\}$ and $d(s) = 0$, $P(u) = s$.

4. For each node $v \in V - S$ find a shortest path P from s to v where all the nodes in $P - v$ are in S (only the last node of P is outside S; v is connected to S by some edge uv).

5. At each step of the algorithm we add $v \in V - S$ into S where $d(v)$ is the smallest.

Computing Shortest Path in Digraphs

6. Once we add v to S, we update the distance of the other nodes outside S because maybe by going to v from s and then to $w \in V - S$, $d(w)$ decreases. We update $P(w)$ (if necessary set $P(w) = v$).

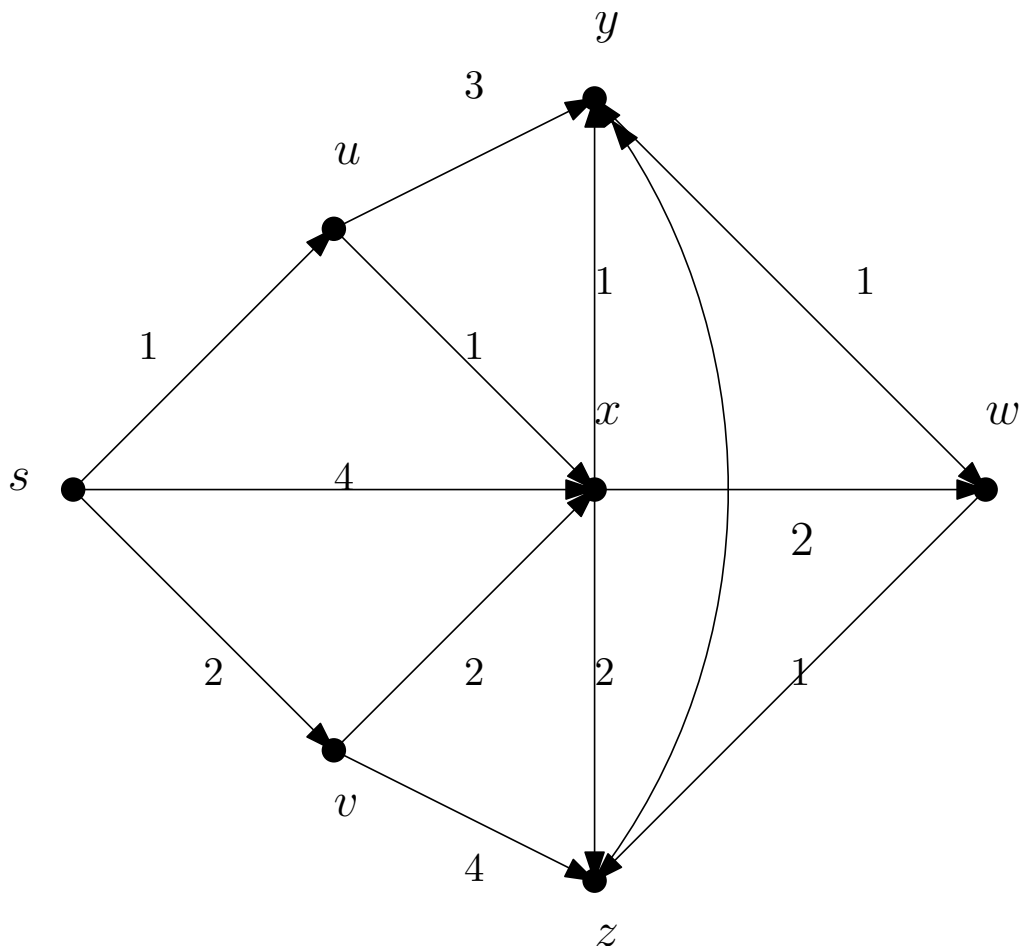
Shortest-Path-Dijkstra(G,A)

```
1. for (i = 1; i <= n; i++) Explore[i]=0;
3. Explore[1]=1;
4. for (i = 2, i <= n; i++) { d[i]=A[1][i];    P[i]=1;}
6. for (i = 2; i <= n; i++)
7. {
8.     int Min = ∞ ;    int index;
9.     for (j = 2; j <= n; j++)
10.        if( Explore[j] ==0 && d[j] < Min )
11.          { Min=d[j]; index=j; }
14.    Explore[index]=1;
15.    for (j = 2, j <= n; j++)
17.        if(Explore[j] ==0 && d[j] > d[index] + A[index][j] )
18.          { d[j]= d[index]+A[index][j];    P[j]=index; }
20.}
```

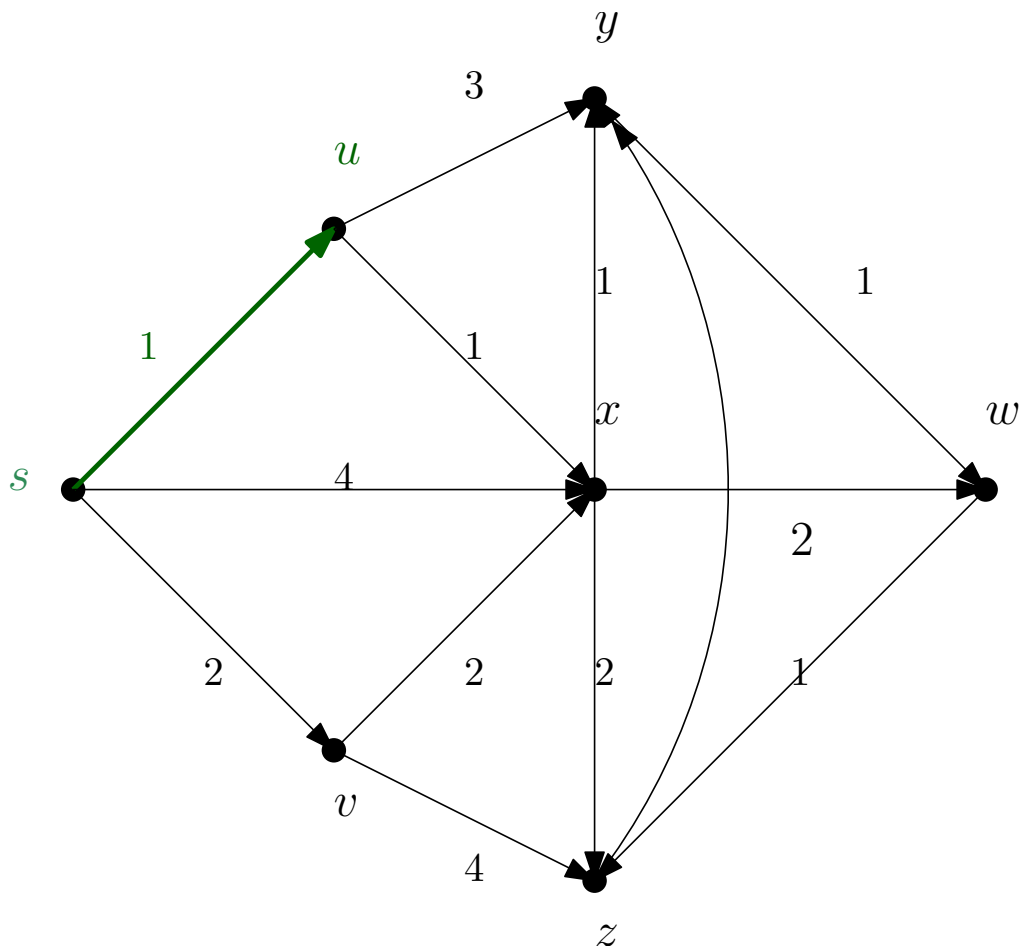
Finding the shortest path

Find-Path(P,u) {

```
1. if (u == s)
2. {
3.       cout << s << endl;
3.       return;
4. }
5. Find-Path(P,P[u]);
6. cout << u << endl;
}
```

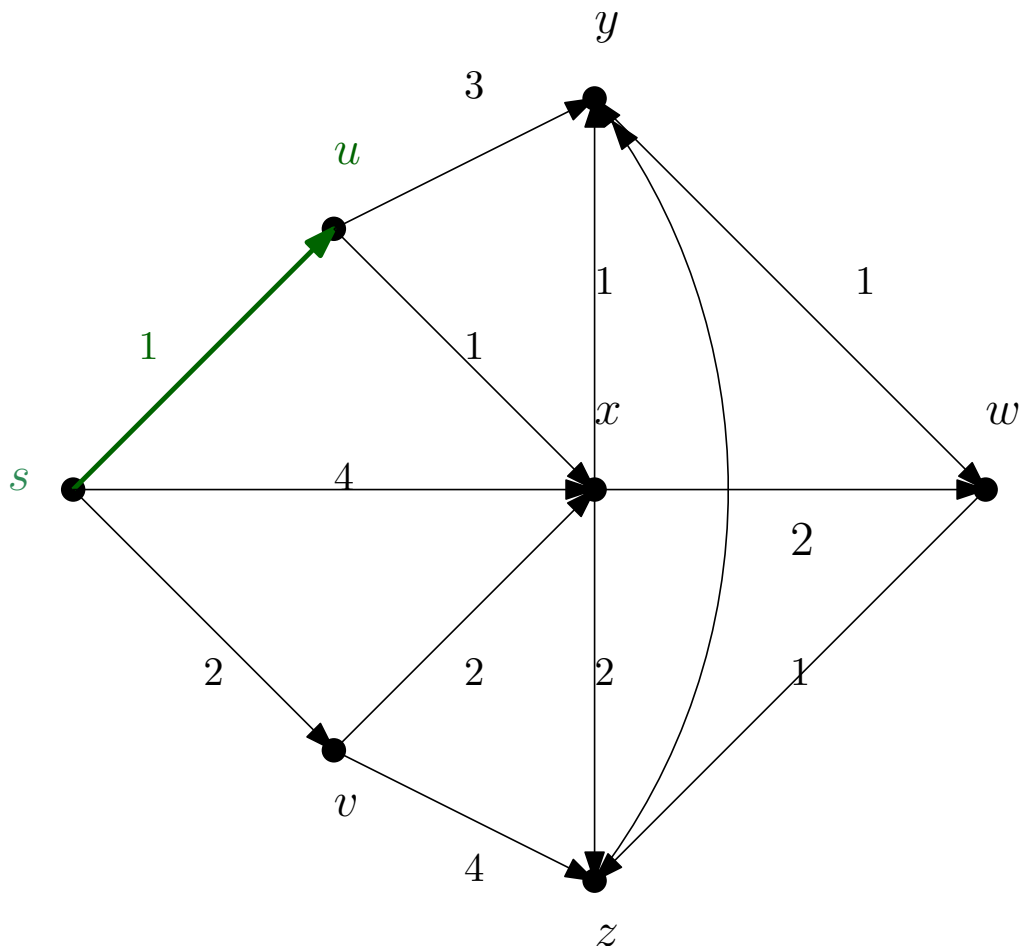


$d(s) = 0,\ d(u) = 1,\ d(v) = 2,\ d(x) = 4,$
$d(y) = d(z) = d(w) = \infty.$



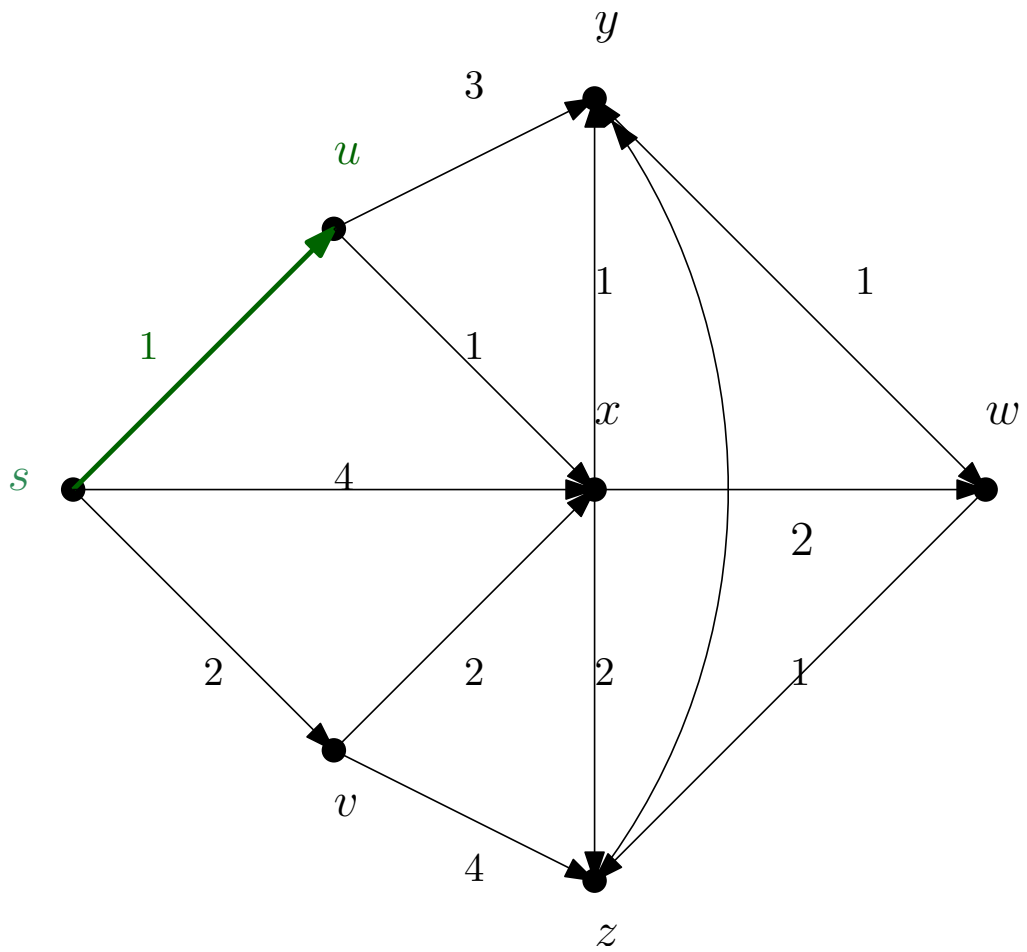
$d(s) = 0,\ d(u) = 1,\ d(v) = 2,\ d(x) = 4,$
$d(y) = d(z) = d(w) = \infty.$

Index $= u$

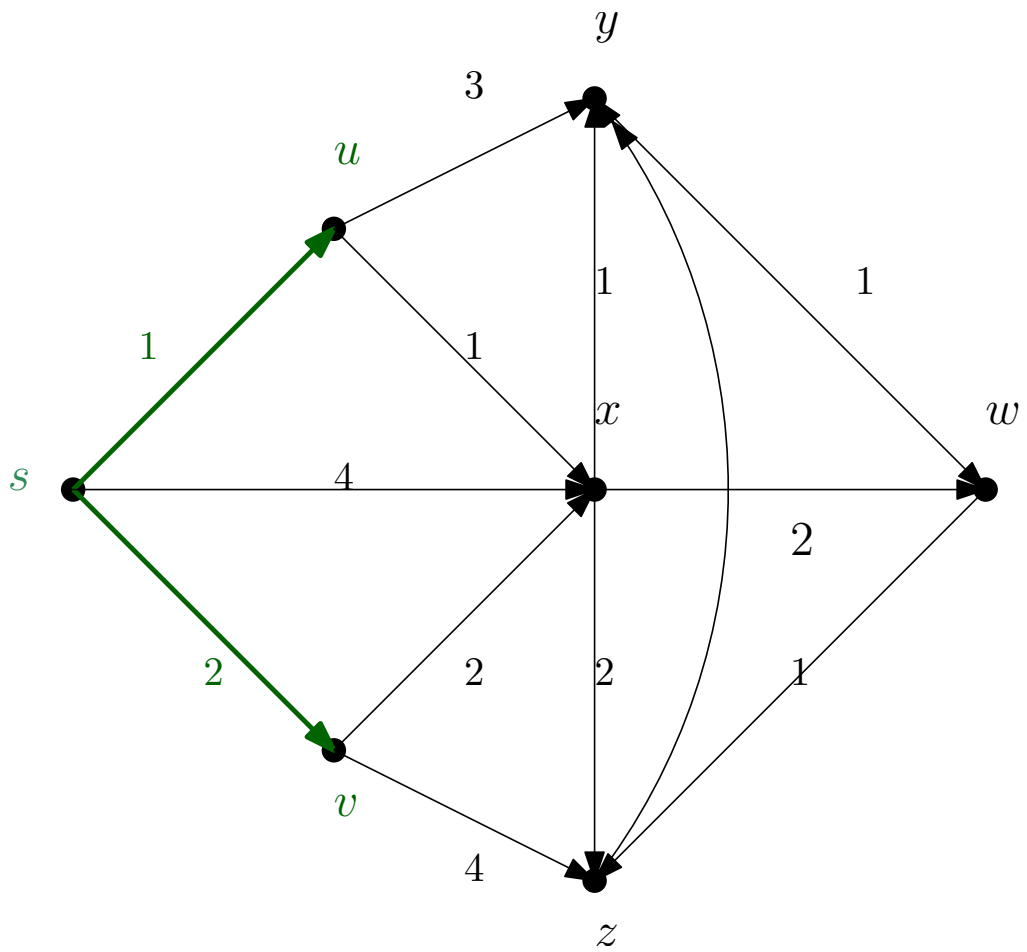


$d(s) = 0,\ d(u) = 1,\ d(v) = 2,\ d(x) = 4,$
$d(y) = d(z) = d(w) = \infty.$

Index $= u$ $d(x) > d(u) + A[u][v]$ then $d(x) = 2,\ d(y) = 4,$

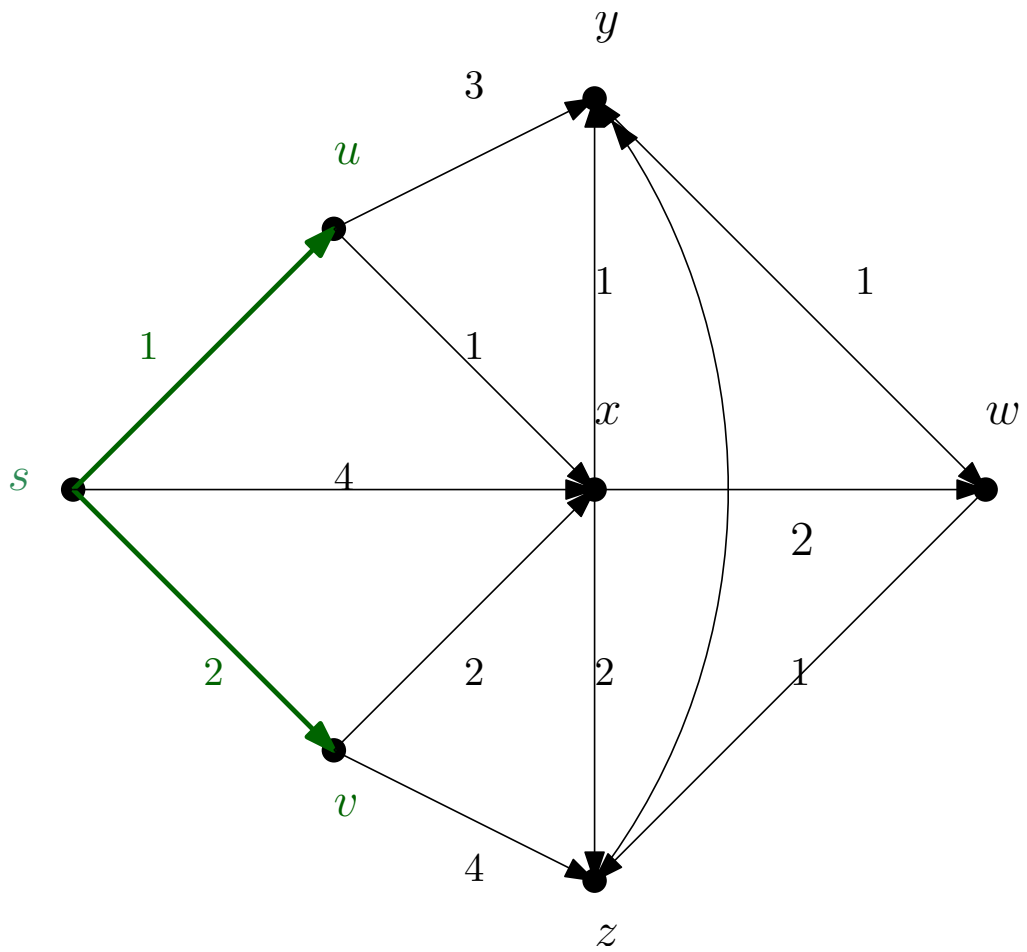


$d(s) = 0$, $d(u) = 1$, $d(v) = 2$, $d(x) = 2$, $d(y) = 4$,
$d(z) = d(w) = \infty$.



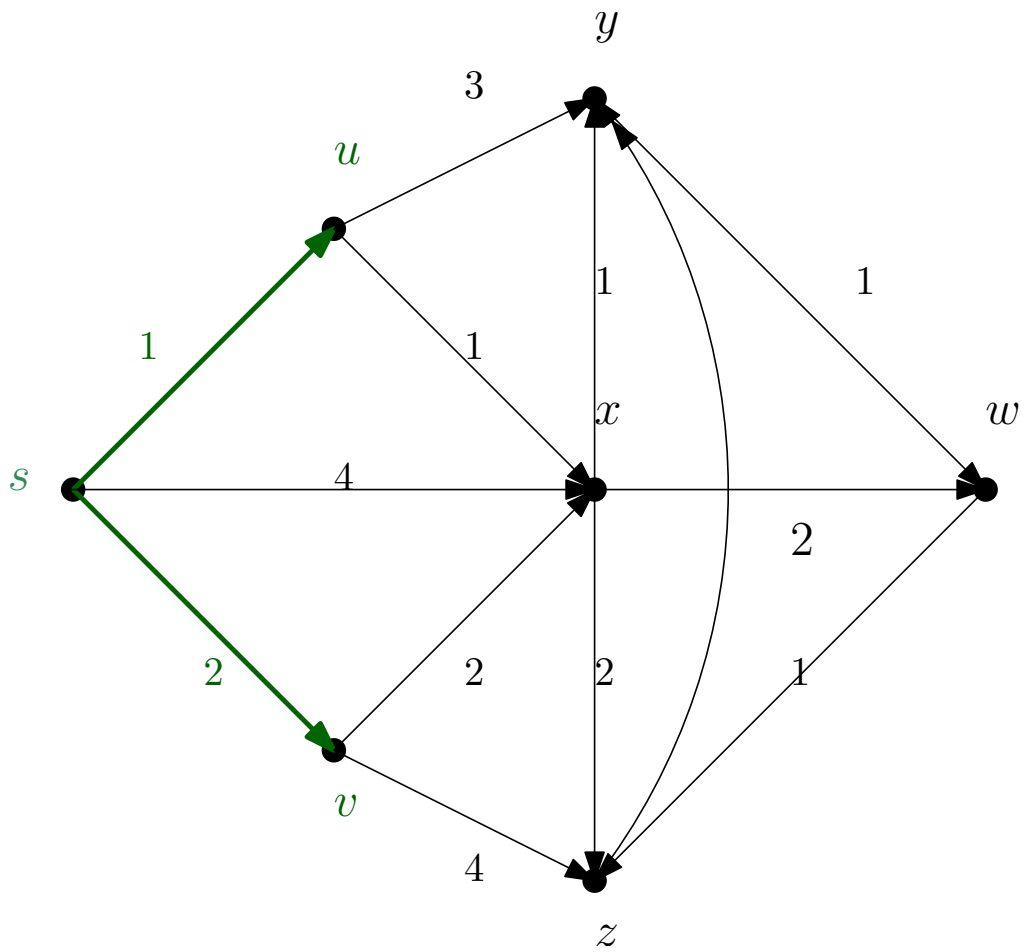
$d(s) = 0$, $d(u) = 1$, $d(v) = 2$, $d(x) = 2$, $d(y) = 4$,
$d(z) = d(w) = \infty$.

Index $= v$,

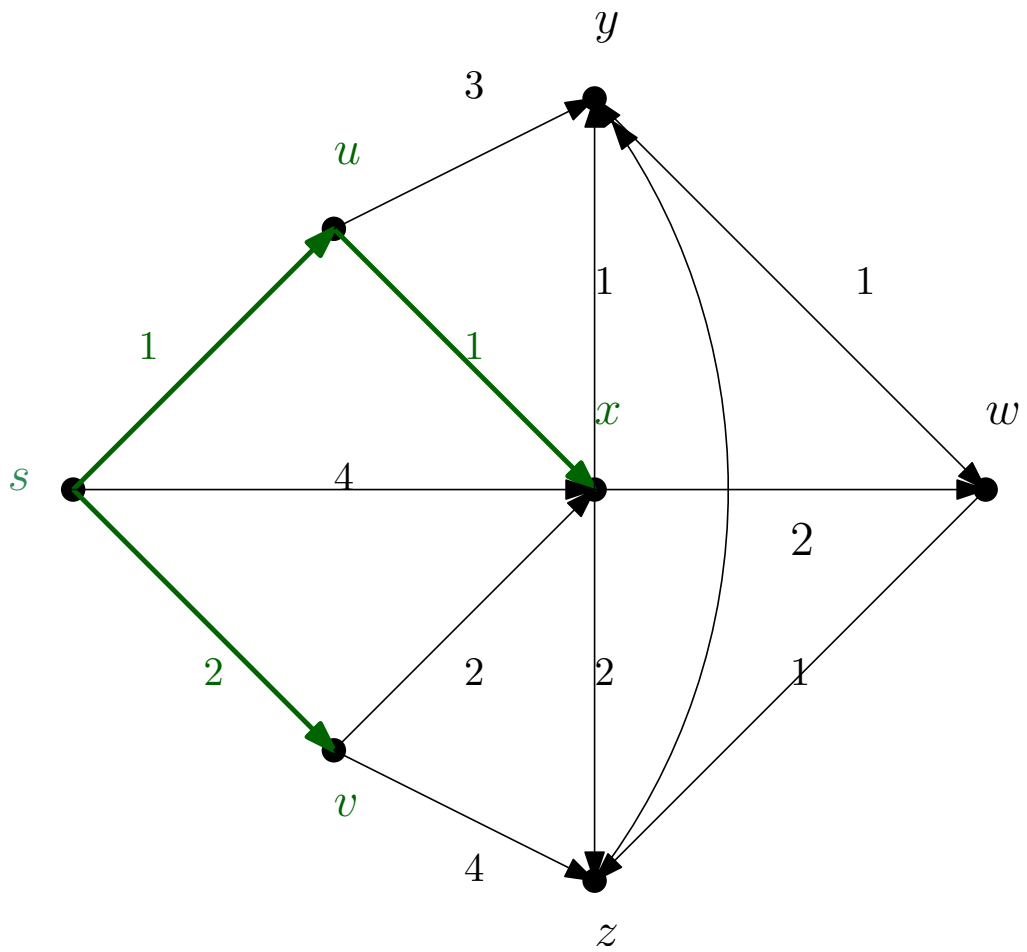


$d(s) = 0$, $d(u) = 1$, $d(v) = 2$, $d(x) = 2$, $d(y) = 4$,
$d(z) = d(w) = \infty$.

Index $= v$, $d(z) > d(v) + A[v][z]$ then $d(z) = 6$,

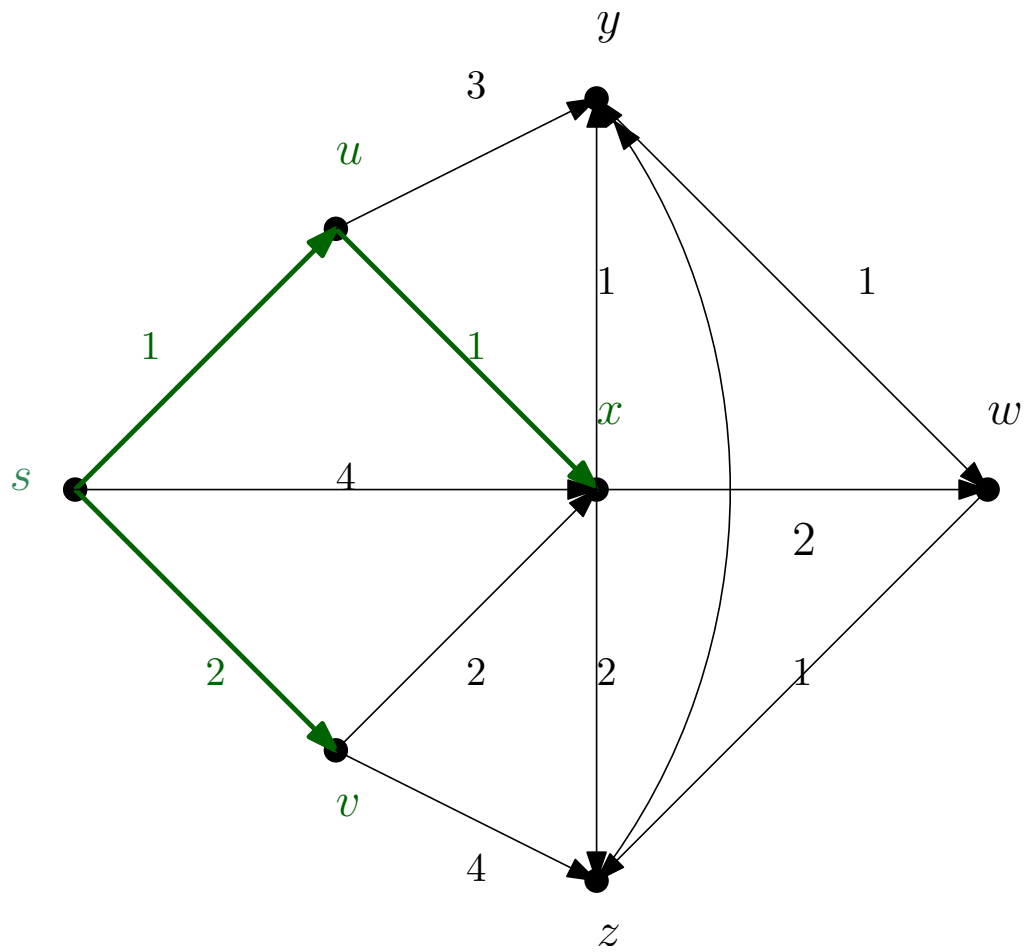


$d(s) = 0$, $d(u) = 1$, $d(v) = 2$, $d(x) = 2$, $d(y) = 4$, $d(z) = 6$, $d(w) = \infty$.



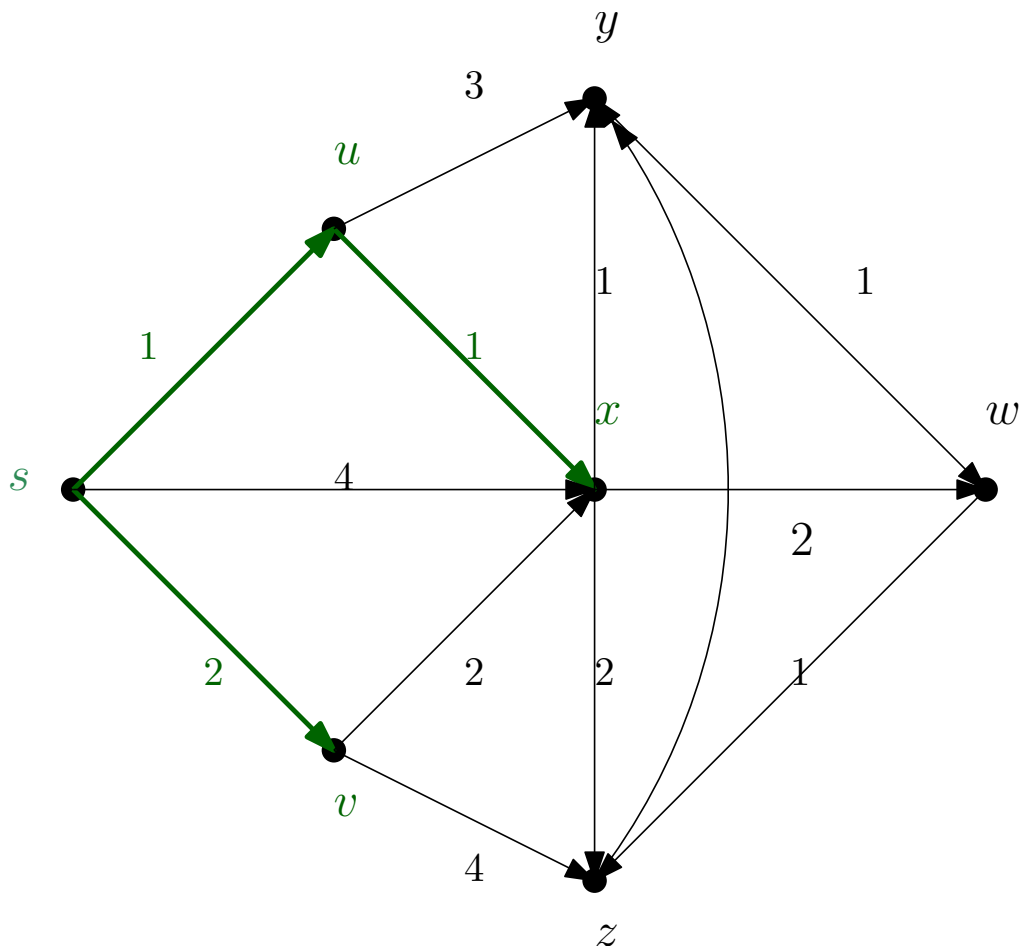
$d(s) = 0$, $d(u) = 1$, $d(v) = 2$, $d(x) = 2$, $d(y) = 4$, $d(z) = 6$, $d(w) = \infty$.

Index $= x$,

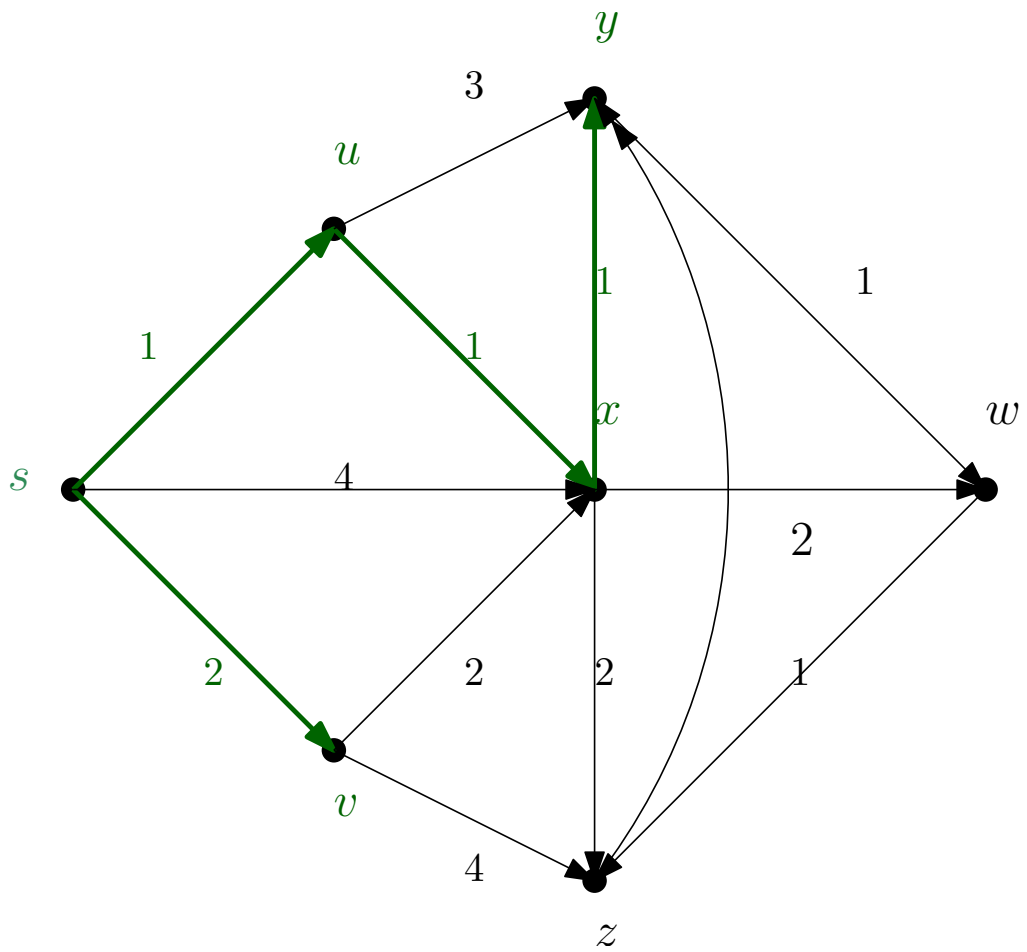


$d(s) = 0$, $d(u) = 1$, $d(v) = 2$, $d(x) = 2$, $d(y) = 4$, $d(z) = 6$, $d(w) = \infty$.

Index $= x$, $d(w) > d(x) + A[x][w]$ then $d(w) = 4$, $d(y) = 3$, $d(z) = 4$

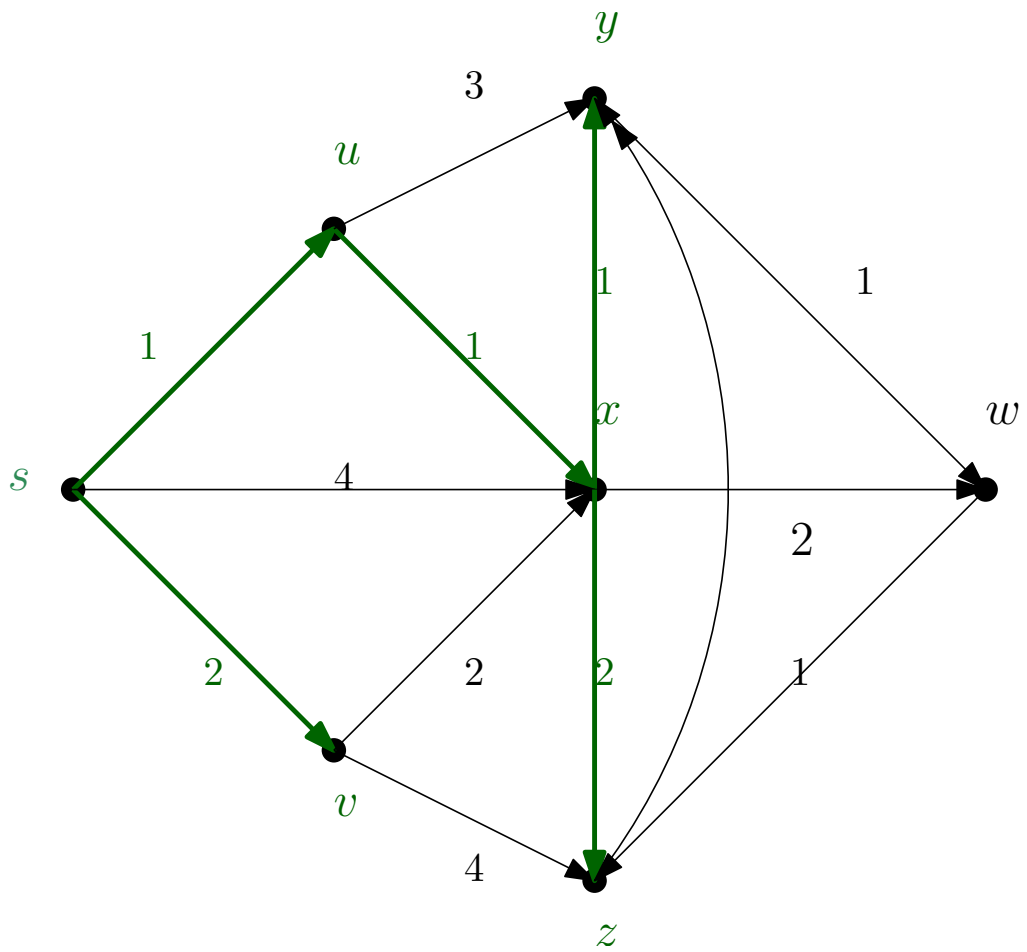


$d(s) = 0$, $d(u) = 1$, $d(v) = 2$, $d(x) = 2$, $d(y) = 3$, $d(z) = 4$, $d(w) = 4$.



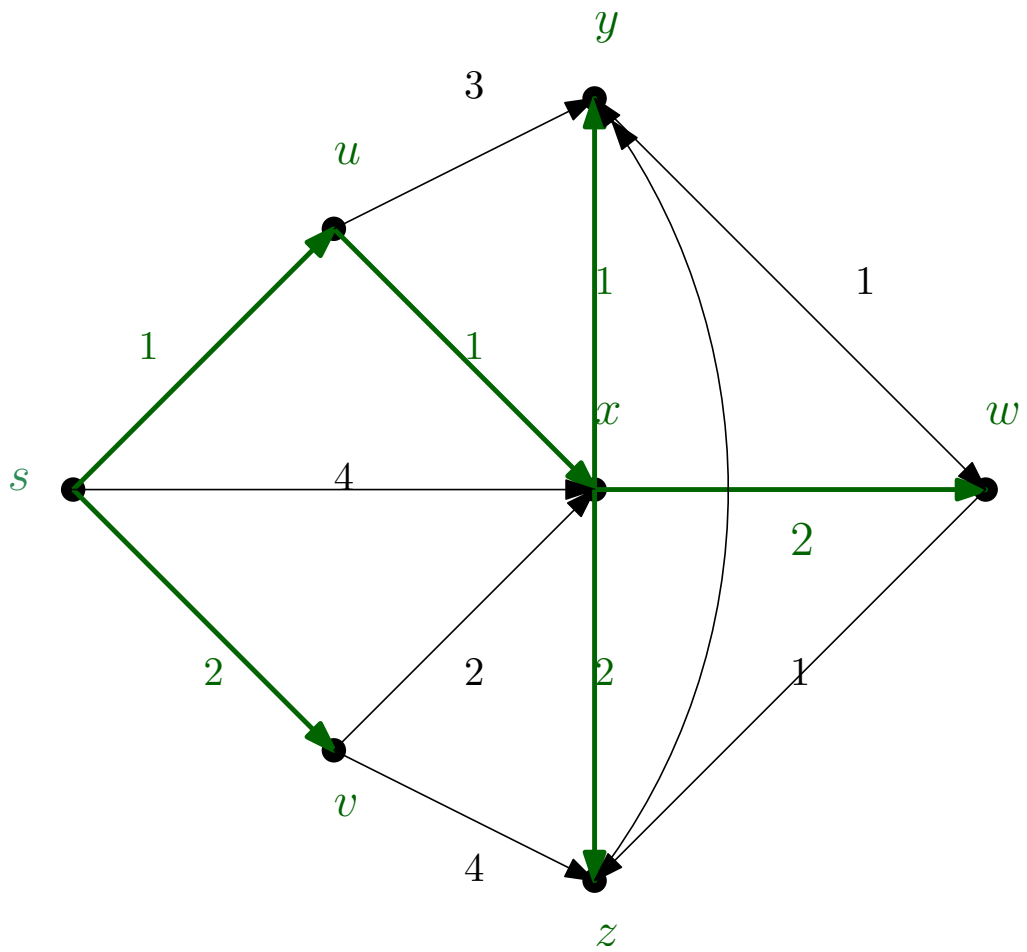
$d(s) = 0$, $d(u) = 1$, $d(v) = 2$, $d(x) = 2$, $d(y) = 3$, $d(z) = 4$, $d(w) = 4$.

Index $= y$,



$d(s) = 0$, $d(u) = 1$, $d(v) = 2$, $d(x) = 2$, $d(y) = 3$, $d(z) = 4$, $d(w) = 4$.

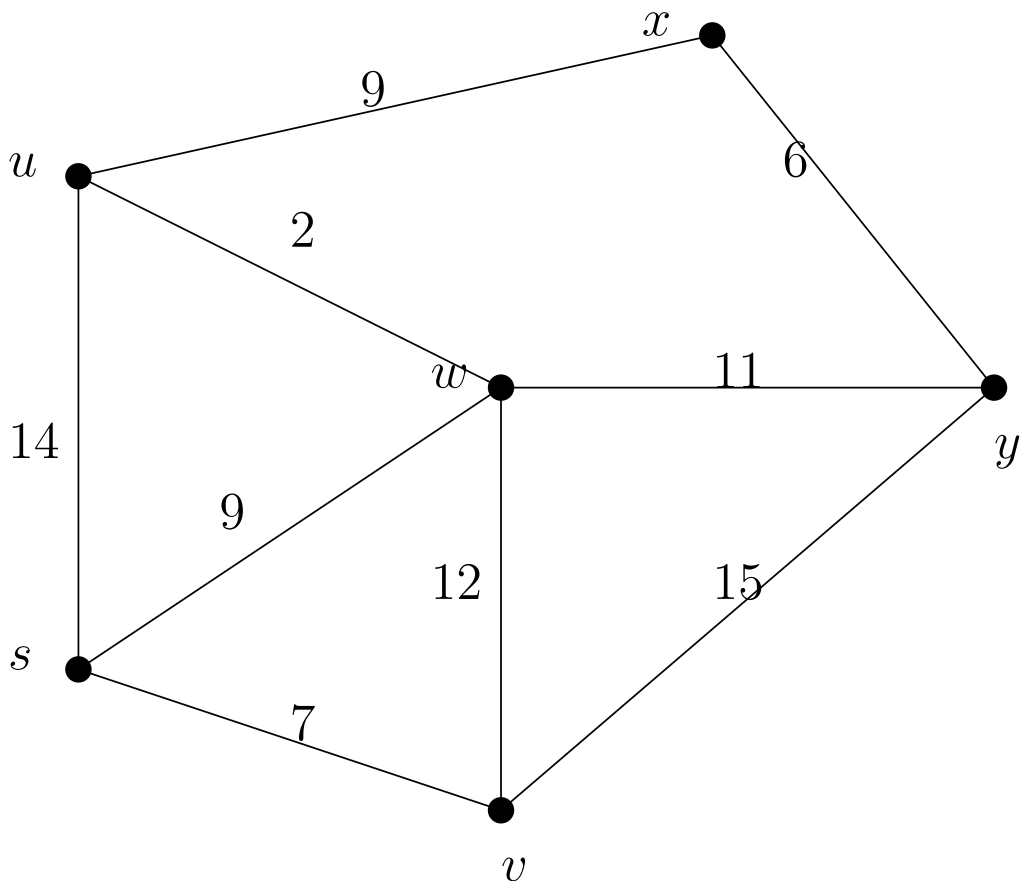
Index $= z$,



$d(s) = 0$, $d(u) = 1$, $d(v) = 2$, $d(x) = 2$, $d(y) = 3$, $d(z) = 4$, $d(w) = 4$.

Index $= w$,

Run the Dijkstra's algorithm on this graph starting from node s.



Lemma

Consider the set S at any point in the algorithm's execution. For each $u \in S$, $d(u)$ is the length of the shortest path from s to u.

Proof : We use induction on the size of S. When S has only s then it is clearly true.

Lemma

Consider the set S at any point in the algorithm's execution. For each $u \in S$, $d(u)$ is the length of the shortest path from s to u.

Proof : We use induction on the size of S. When S has only s then it is clearly true.

Suppose by induction $d(u)$ is the length of the shortest path from s to $u \in S$. We show that for any node v that is placed into S, the algorithm computes $d(v)$ correctly.

Lemma

Consider the set S at any point in the algorithm's execution. For each $u \in S$, $d(u)$ is the length of the shortest path from s to u.

Proof : We use induction on the size of S. When S has only s then it is clearly true.

Suppose by induction $d(u)$ is the length of the shortest path from s to $u \in S$. We show that for any node v that is placed into S, the algorithm computes $d(v)$ correctly.

According to the algorithm v is placed into S at some step and maybe $d(v)$ is updated when u was added into S.

So at that time $d(v) = d(u) + A[u][v]$ or has its previous value(not updated). Suppose there is a path P from s to v that has length less than $d(v)$ (according to the algorithm). Let $P = P_x yQ$ where xyQ is outside S and P_x is inside S.

So at that time $d(v) = d(u) + A[u][v]$ or has its previous value(not updated). Suppose there is a path P from s to v that has length less than $d(v)$ (according to the algorithm). Let $P = P_x y Q$ where xyQ is outside S and P_x is inside S.

This means that the $d(y) \geq d(v)$ because we consider v before y. Since the length of Q is not negative, $\ell(P) = d(y) + \ell(Q) \geq d(v)$ ($\ell(P)$ is the length of P).

Finding a longest path in a DAG

Let D be an acyclic digraph where each arc has a non-negative weight. For every node u of D compute the longest path (longest weighted path) from s to u.

Longest-Path-Algorithm(D, s)

1. Initial queue Q to be empty.
2. For every node v set the $Indegree[v]$ to be the number of nodes having arc to v.
3. For every node u set $\ell d(u) = A[s][u]$.
4. For every node v, If $(Indegree[v] = 0)$ { $Q.add(v)$; }
5. While Q is not empty
6. $\quad u = Q.delete()$;
7. $\quad$ For every arc $uw \in A(D)$
8. $\quad\quad Indegree[w] = Indegree[w] - 1$;
9. $\quad\quad$ If $(\ell d(u) + A[u][w] > \ell d(w))$
10. $\quad\quad\quad \ell d(w) = \ell d(u) + A[u][w]$
11. $\quad\quad$ If $(Indegree[w] = 0)$
12. $\quad\quad\quad Q.add(w)$;

Problem

Dijkstra vs. Prim. *As you may have noticed, Dijkstras and Prims algorithms are very similar. In this question, you will explore their similarities and differences.*

- *(a) Show that running Dijkstras algorithm on an undirected, connected graph G (with non-negative edge costs) will produce a spanning tree of G.*
- *(b) Prove or give a counter-example to the following statement: A tree produced by Dijkstras algorithm on a given (undirected, connected, weighted) graph G is a Minimum Spanning Tree.*

Problem

Write a pseudo code for finding a longest weighted path between two given nodes u, v in an acyclic digraph D.

Problem

Show that if a tournament is NOT acyclic (it has a directed cycle) then it has a directed cycle of length three.

Problem

Write a program that gets a tournament T of size n as input, and it produces an ordering $v_1, v_2, \ldots, v_n$ of the nodes such that $v_i \to v_{i+1}$, (for every $1 \leq i \leq n-1$).

Problem

Write a program that gets a strongly connected tournament T of size n as input, and it produces an ordering $v_1, v_2, \ldots, v_n, v_1$ of the nodes such that $v_i \to v_{i+1}$, (for every $1 \leq i \leq n$, $v_{n+1} = v_1$).

Problem

We are given a digraph G. Design an algorithm to find the girth of G. The girth of G is the length (number of arcs) of a shortest (directed) cycle in G.