Minimum Weight Spanning Trees in Weighted Graphs

A *weighted* graph G is one in which each edge e is associated a nonnegative real number $w(e)$, called the weight of e. If H is a subgraph of G, then the weight of H is defined as:

$$\sum_{e \in E(H)} w(e) .$$

Given a connected weighted graph G,

how to find a connected subgraph with minimum weight?

that is, how to find a spanning tree of G with minimum weight?

The following *tree-search algorithm* is due to Jarník (1930) and Prim (1957).

J-P Algorithm

1. Take any vertex, say v_1, as the root, and set $T_1 = v_1$;
2. For $k \geq 1$, choose an edge xy with $x \in V(T_k)$ and $y \in V(G) \backslash V(T_k)$ such that $w(xy)$ is minimum among all edges between $V(T_k)$ and $V(G) \backslash V(T_k)$; Set $T_{k+1} = T_k + xy$.

The above J-P Algorithm outputs a tree, called *J-P tree*.

The following theorem shows that the J-P algorithm runs correctly.

Theorem 11. Every J-P tree is an optimal tree.

Proof. Let T be a J-P tree with root r.

We will prove that T is an optimal tree by induction on $v(G)$.

Assume that the first edge added to T is an edge e of least weight in the edges incident with r; in other words, $w(e) \leq w(e')$ for all edges e' incident with r. To begin with, we show that some optimal tree includes this edge e.

Let T' be an optimal tree. We may assume that $e \notin E(T')$. Clearly, $T' + e$ has a unique cycle C. Let e' be the other edge of C incident with r. Then $T'' = (T' + e) \backslash e'$ is a spanning tree of G. Moreover, because $w(e) \leq w(e')$,

$$w(T'') = w(T') + w(e) - w(e') \leq w(T').$$

Since T' is an optimal tree, so is T''. Moreover, T'' contains e.

Let $G^* = G/e$ and r^* the vertex resulting from the contraction of e.

It is not difficult to see there is a one-to-one correspondence between the set of spanning trees of G that contain e and the set of all spanning trees of G^*.

Let $T^* = T/e$. To show T is an optimal tree of G, it suffices to show T^* is an optimal tree of G^* with root r^*.

Note that $\partial(T_{i+1}) = \partial(T_i^*)$ for $1 \le i \le n-2$,

we can see that an edge of minimum weight in $\partial(T_{i+1})$ is also an edge of minimum weight in $\partial(T_i^*)$.

Because the final tree T is a J-P tree of G, we deduce that the final tree T^* is a J-P tree of G^* with root r^*.

As $v(G^*) < v(G)$, by induction hypothesis, T^* is an optimal tree of G^*.

Recall there is an optimal tree containing the edge e as shown before,

we conclude that T is an optimal tree of G.

The Shortest Path in Weighted Graphs

For a connected weighted graph G, the *distance* between two given vertices u and v is the weight of (u,v)-path with minimum weight, write as $d(u,v)$. For $S \subset V(G)$, $d(u,S) = \min\left\{d(u,v) : v \in S\right\}$.

For two given vertices u_0 and v_0, how to find a shortest (u_0, v_0)-path? Assume that $S \subset V(G)$, $u_0 \in S$ and $d(u_0, u)$ has been known for any $u \in S$. Set $V(G) \backslash S = S'$. If $P = u_0 \cdots u'v'$ is a shortest from u_0 to S', then $u' \in S$ and $P_1 = u_0 \cdots u'$ is a shortest (u_0, u')-path. Thus,

$$d(u_0, v') = d(u_0, u') + w(u'v'),$$

and

$$d(u_0, S') = \min_{u \in S, v \in S'} \left\{d(u_0, u) + w(uv)\right\}.$$

Let $S_0 = \{u_0\}$. Choose $u_1 \in S_0'$ such that

$$d(u_0, u_1) = d(u_0, S_0'),$$

$$d(u_0, S_0') = \min_{u \in S_0, v \in S_0'} \left\{ d(u_0, u) + w(uv) \right\} = \min_{v \in S_0'} \left\{ w(u_0 v) \right\}.$$

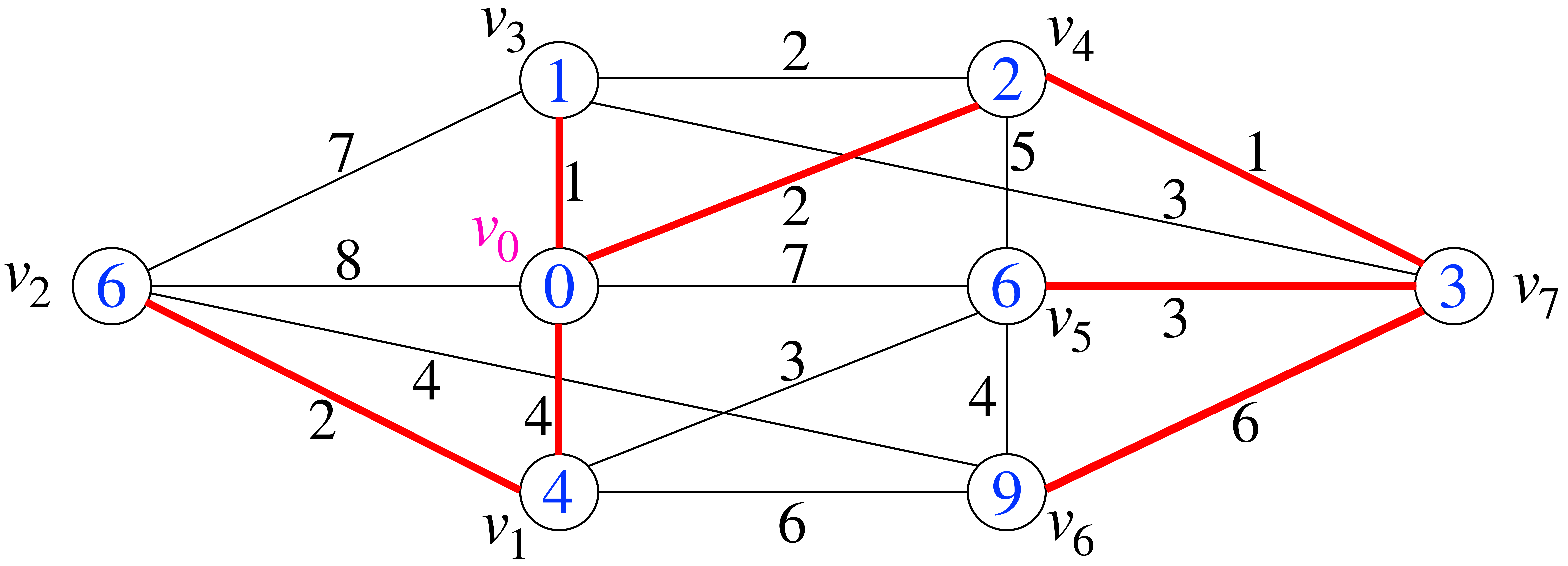
Set $S_1 = \{u_0, u_1\}$ and $P_1 = u_0 u_1$.
Assume that $S_k = \{u_0, u_1, \ldots, u_k\}$ and the corresponding shortest paths are $P_1, P_2, \ldots, P_k$. Then by

$$d(u_0, S_k') = \min_{u \in S_k, v \in S_k'} \left\{ d(u_0, u) + w(uv) \right\},$$

we can choose $u_{k+1} \in S_k'$ such that

$$d(u_0, u_{k+1}) = d(u_0, S_k').$$

Set $S_{k+1} = S_k \cup \{u_{k+1}\}$.
Assume that $d(u_0, u_{k+1}) = d(u_0, u_j) + w(u_j u_{k+1})$. Just let $P_{k+1} = u_0 P_j u_j u_{k+1}$.



Dijkstra Algorithm

1. Let $\ell(u_0) = 0, \ell(v) = \infty$ for any $v \neq u_0$. Set $S_0 = \{u_0\}$ and $i = 0$.
2. For any $v \in S_i'$, replace $\ell(v)$ with $\min_{u \in S_i} \left\{ \ell(v), \ell(u) + w(uv) \right\}$. Calculate

$$\min_{v \in S_i'} \left\{ \ell(v) \right\}$$

   and denote the vertex v reach the minimum as u_{i+1}. Set

$$S_{i+1} = S_i \cup \left\{ u_{i+1} \right\}.$$

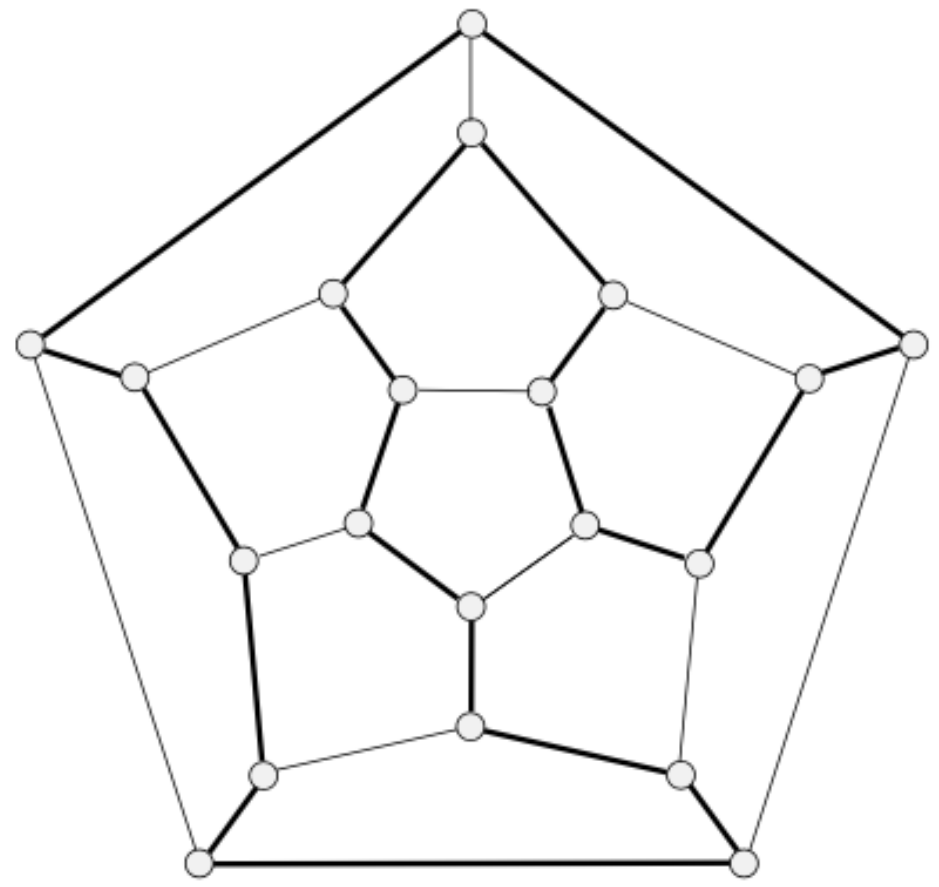
3. The algorithm stops when $i = n - 1$.

   If $i < n - 1$, then replace i with $i + 1$, go to Step 2.

Hamilton Problems

A path or cycle in a graph which contains every vertex of the graph is called a *Hamilton path* or *Hamilton cycle* of the graph.

Such paths and cycles are named after Sir William Rowan Hamilton, who described, in a letter to his friend Graves in 1856, a mathematical game on the dodecahedron in which one person sticks pins in any five consecutive vertices and the other is required to complete the path so formed to a spanning cycle.



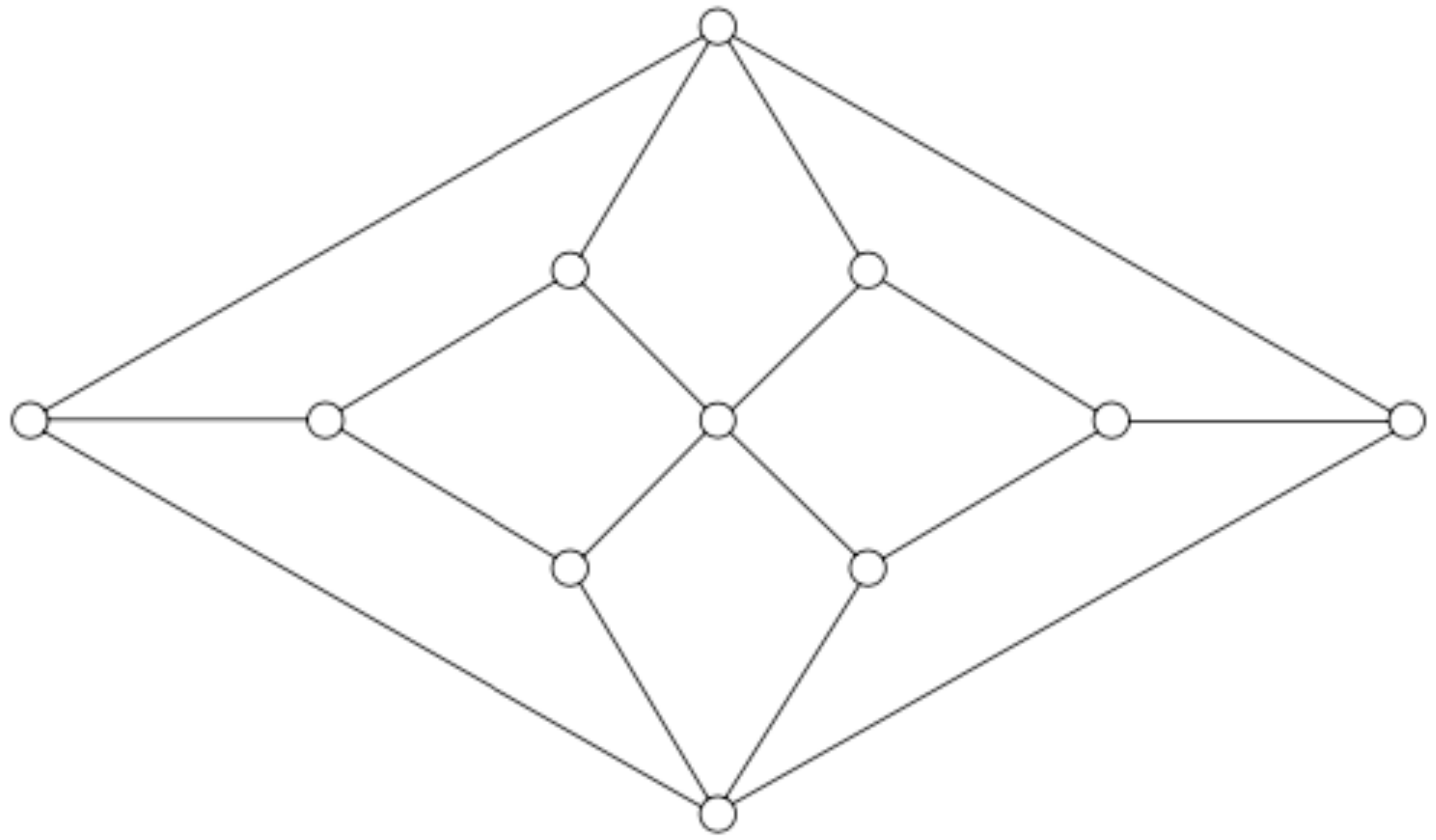
Let G be a graph and $S \subset V(G)$.

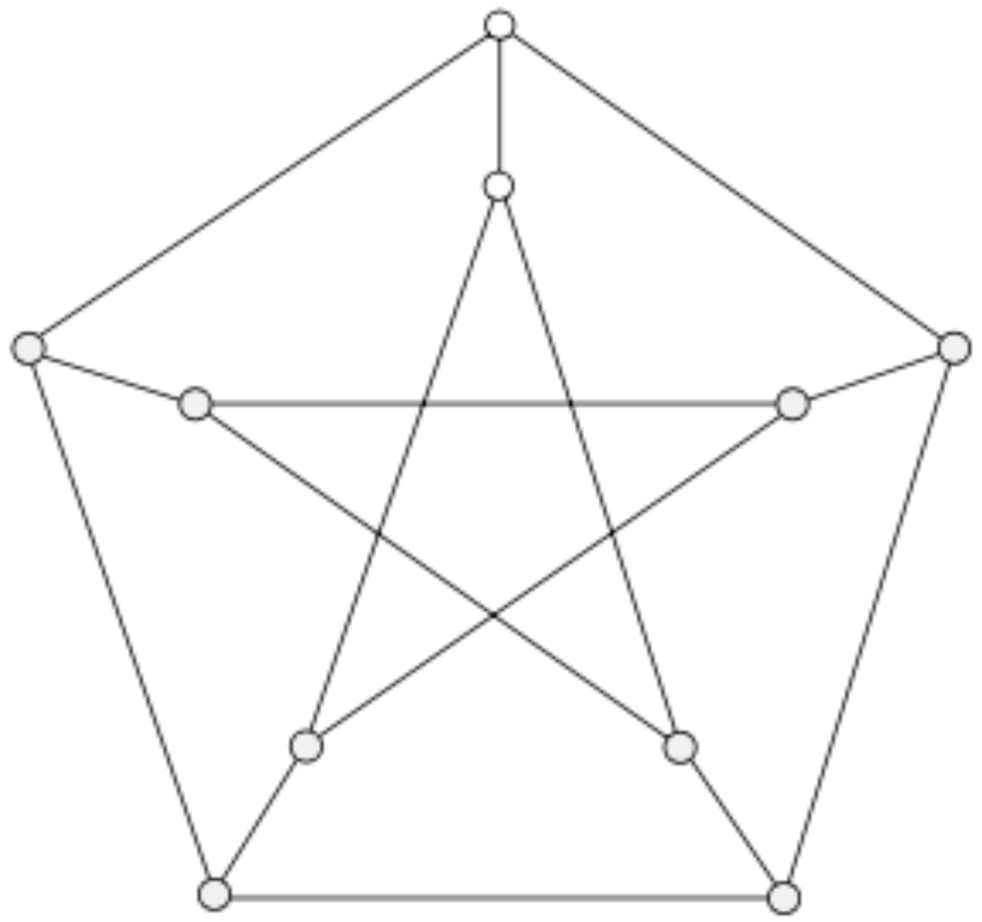
We use $\omega(G-S)$ to denote the number of components of $G-S$.

Theorem 12. Let G be a graph with a Hamilton cycle. Then for any $S \subset V(G)$,

$$\omega(G-S) \leq |S|.$$

Theorem 12 is a necessary condition for a graph to be hamiltonian.

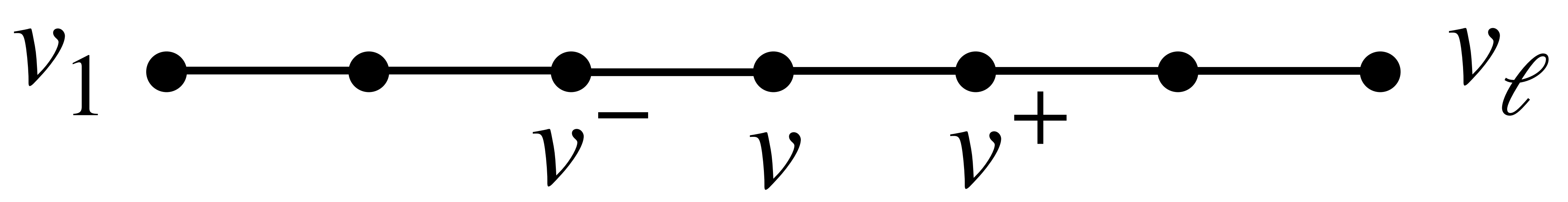




Theorem 13(Dirac). Let G be a simple graph of order $n \geq 3$. If $\delta(G) \geq n/2$, then G is Hamiltonian.

Proof. It is not difficult to show that G is connected (in fact, 2-connected).

Let $P = v_1 v_2 \cdots v_\ell$ be a longest path in G. For any $v \in V(P)$, let v^-, v^+ be the predecessor and successor of v, respectively.

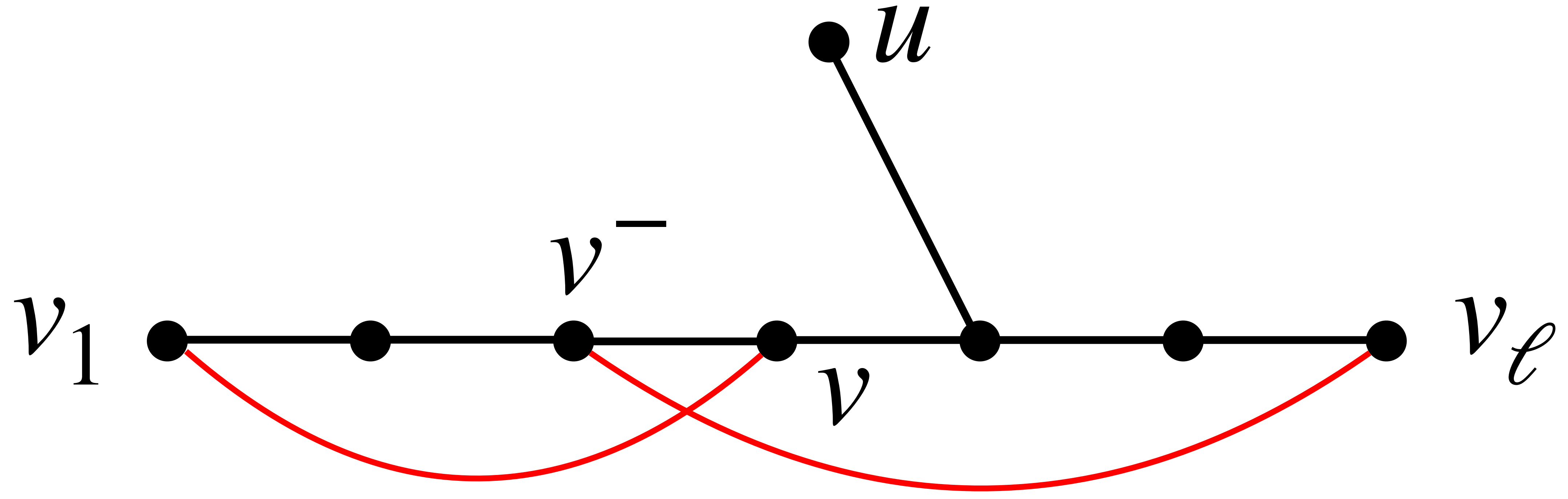


Let

$$N_P^+(v_\ell) = \left\{ v^+ : v \in N_P(v_\ell) \right\}.$$

It is not difficult to see that $N_P^+(v_\ell) \subseteq V(P)$.

If $N_P(v_1) \cap N_P^+(v_\ell) \neq \varnothing,$ say $v \in N_P(v_1) \cap N_P^+(v_\ell),$ then $C = v_1 \overrightarrow{P} v^- v_\ell \overleftarrow{P} v v_1$ must be a Hamiltonian cycle, for otherwise,



If $N_P(v_1) \cap N_P^+(v_\ell) = \varnothing,$ then since $\delta(G) \geq n/2$, we have

$$|P| \geq |N_P(v_1)| + |N_P^+(v_\ell)| + 1 \geq n/2 + n/2 + 1 = n + 1,$$

which is impossible.

Exercise 3.

1. Use Dijkstra algorithm, compute the distance $d(v_5, v_i)$ for any other v_i.