1. Reading: K. Rosen *Discrete Mathematics and Its Applications*, 7.6

2. The main message of this lecture:

**Finding the shortest path in a graph is based on an elementary and thus universal principle: every segment of an optimal process is also optimal.**

**Definition 32.1.** Yet another type of graphs: **weighted graph** is a graph (simple, multi..., pseudo..., directed) with a number assigned to each edge. The length of a path in a weighted graph is the sum of the weights of the edges of this path. The **shortest path** between two given vertices is the path of least length between them.

We may assume that the shortest path is a graph never visits the same vertex twice, since otherwise one could make this path even shorter by skipping the cycle. A raw upper bound for the number of all possible paths between two given vertices is about \( n! \), where \( n \) is a total number of vertices in a graph. For example, in the complete graph \( K_n \) the number of Hamilton paths only between two distinct vertices is \( (n - 2)! \). Indeed, there are \( n - 2 \) choices for the first step, \( n - 3 \) for the second, etc. Therefore, the exhaustive search of all possible paths and comparing its lengths cannot be regarded as a general algorithm. However, using a very basic observation that every initial segment from \( a \) to \( b \) of the shortest path from \( a \) to \( c \) is itself the shortest path (between \( a \) and \( b \)) we can reduce the search dramatically.

**Definition 32.2.** The following **Dijkstra’s algorithm** finds the length of a shortest path in a connected simple weighted graph \( G \) with vertices \( v_1, v_2, \ldots, v_n \), and weights \( w(u, v) \) between vertices \( u \) and \( v \). If \( u \) and \( v \) are not connected then \( w(u, v) = \infty \). The algorithm relies on a series of iterations of adding a labeled vertex to a set \( S \) of distinguished vertices. The labels of a vertices in \( S \) do not change and they represent the shortest distance between a given vertex and the origin \( a \). Labels of vertices outside \( S \) are recalculated in every loop. The algorithm terminates when the target vertex \( z \) is captured by \( S \).

\[
\begin{align*}
\text{for } i & \ := \ 1 \ \text{to} \ n \\
L(v_i) & = \infty \\
L(a) & = 0 \\
S & = \emptyset \\
\text{while } z & \not\in S \\
\text{begin} \\
\quad u & := \text{a vertex not in } S \text{ with the minimal label} \\
\quad S & := S \cup \{u\} \\
\quad \text{for all vertices } v \not\in S \\
\quad \quad \text{if } L(u) + w(u, v) < L(v) \ \text{then } L(v) := L(u) + w(u, v) \\
\text{end} \\
\end{align*}
\]

(\( L(z) = \text{length of shortest path from } a \ \text{to } z \)).

**Example 32.3.** Cf. lecture slides and the book.

**Theorem 32.3.** Dijkstra’s algorithm finds the length of a shortest path between two vertices in a connected simple undirected graph.
**Proof.** By induction on the number of iterations made we prove the following assertion $A(k)$: “after $k$ iterations the set $S = S_k$ satisfies

1) the label of $v \in S_k$ is the length of the shortest path from $a$ to $v$

2) the label of $v \notin S_k$ is the length of the shortest path from $a$ to $v$ that contains only (besides $v$) vertices in $S_k$.

**BASE.** $k = 0$. i.e. before any iteration is carry out. Then both 1) and 2) hold since $S_0 = \emptyset$ and there is no vertices in $S_0$ (covers 1.) and no paths in $S_0$ (covers 2.).

**INDUCTION HYPOTHESIS.** After $k$ iterations both 1) and 2) hold.

**INDUCTION STEP.** Let $u$ be a vertex added to $S_k$ at the $(k + 1)$st iteration. This means $u$ has the least label among vertices not in $S_k$. We have to establish that both 1) and 2) hold for $S_{k+1}$.

For 1) it suffices to check $u$ since all the old labels in $S_k$ have not changed. Suppose the opposite, i.e. that there is a path $P$ from $a$ to $u$ shorter than $L(u)$. This path $P$ cannot be in $S_k$ only, since then, by the I.H. 2) and by the choice of $u$, this $u$ is the closest to $a$. The path $P$ cannot contain vertices not from $S_k$ either, since the first such vertex in $P$ would have a lesser label than $u$ and would be added to $S_k$ instead.

Checking 2). Pick any $x \notin S_{k+1}$, and consider the shortest path $P$ in $S_{k+1}$ from $a$ to $x$. There are two possibilities. Case A: this path $P$ does not contain $u$. Then, by the I.H. 1), $P$ is the shortest path in $S_k$ from $a$ to $x$. By the description of step $k + 1$, the label $L(x)$ has not changed, thus $L(x)$ remains the length of the shortest path in $S_{k+1}$ from $a$ to $x$. Case B. The shortest path $P$ from $a$ to $x$ in $S_{k+1}$ contains $u$. Then $P$ consists of the interval from $a$ to $u$ of the shortest possible length followed by the edge from $u$ to $x$ (show why $P$ cannot make any more steps between $u$ and $x$!). Then the length of $P$ is $L(u) + w(u, x)$, which is exactly the $L(x)$ after $(k + 1)$st iteration.

**Theorem 32.4.** *The computational complexity of Dijkstra’s algorithm is $O(n^2)$ comparisons and additions, where $n$ is the number of vertices in the original graph.*

**Proof.** Number of iterations $n - 1$. Identifying the vertex not in $S$ with the smallest label takes $n - 1$ comparisons. Updating the label of each vertex not in $S$ takes $2(n - 1)$ additions and comparisons, thus the total number of additions and comparisons in each iteration is not more then $3(n - 1)$.

**Definition 32.5.** The traveling salesman problem is to find the shortest Hamilton circuit in a weighted, complete undirected graph.

This is a famous $NP$-complete problem.

**Homework assignments.** (The third installment due Friday 04/20)

32A:Rosen7.6-8d; 32B:Rosen7.6-14(Miami-LA); 32C:Rosen7.6-18.