1 Shortest path

1.1 Non-negative lengths

We are given a digraph $D = ([n], E)$ with vertex set $[n]$. Let $\mathcal{P}$ denote the set of paths in $D$ and let $\ell : \mathcal{P} \rightarrow \mathbb{R}$. Think initially that there are edge lengths $\ell : E \rightarrow \mathbb{R}_+$ and that

$$\ell(P) = \ell_{\text{reg}}(P) = \sum_{e \in P} \ell(e).$$

Dijkstra’s Algorithm:

\begin{verbatim}
begin
for $i = 2, \ldots, n$, $d(i) \leftarrow \ell(1,i)$, $P_i \leftarrow (1,i)$; $S_1 \leftarrow \{1\}$
for $k = 2, \ldots, n$ do;
    begin
        $d(i) = \min \{d(j) : j \notin S_k\}$;
        $S_{k+1} \leftarrow S_k \cup \{i\}$;
        for $j \notin S_{k+1}$ do
            if $d(j) > \ell(P_i, j)$ then $d(j) \leftarrow \ell(P_i, j)$, $P_j \leftarrow (P_i, j)$;
    end
end
\end{verbatim}

Lemma 1.1. On termination of Dijkstra’s Algorithm, $d(i) = \ell(P_i)$ is the minimum length of a path from 1 to $i$, for all $i$.

Proof. At each stage we can verify by induction on $k$ that for each $i \notin S_k$, $d(i)$ is the minimum length of a path from 1 to $i$ for which all vertices but $i$ are in $S_k$. If true for $k$ then when we add vertex $i$ we simply update the $d$’s correctly.
Suppose that $i$ is added at Step $r$. Let $P = (x_0 = 1, x_2, \ldots, x_m = i)$ be a path from 1 to $i$. Suppose that $x_0, x_1, \ldots, x_{l-1} \in S_{r-1}$ and $x_l \notin S_{r-1}$. Then,

$$\ell(P) \geq \ell(x_0, x_1, \ldots, x_l) \geq d(x_l) \geq d(i) = \ell(P_i).$$

Note now that all we have assumed about $\ell$ is that $P = (P_1, P_2)$ implies $\ell(P_1) \leq \ell(P)$. (1)

In which case, we can apply the algorithm to solve problems where path length is defined as follows:

**Time dependent path lengths:** Suppose edge $e = (x, y)$ has two parameters $a_e, b_e \geq 0$ and that if we start a walk at time 0 and arrive at $x$ at time $t$ then the edge length is $a_e + b_e t$. Suppose that $P = (e_0, e_1, \ldots, e_k)$ as a sequence of edges and that $P_i = (e_0, e_1, \ldots, e_i)$. Then we now have $\ell(P_0) = a_{e_0}$ and $\ell(P_i) = a_{e_i} + b_{e_i} \ell(P_{i-1})$.

**Visit $S$ in a fixed order:** $S$ is a set of vertices and feasible paths must visit $S$ in some fixed order. Individual edge lengths are non-negative. Then

$$\ell(P) = \begin{cases} 
\ell_{\text{reg}}(P) & P \cap S \text{ visited in correct order.} \\
\infty & \text{Otherwise.}
\end{cases}$$

**Avoid $S$:** $S$ is a set of vertices and there is a penalty of $f(k)$ for visiting $S$, $k$ times. Here $f(k)$ is monotone increasing in $k$. Individual edge lengths are non-negative. Then

$$\ell(P) = \ell_{\text{reg}}(P) + f(|V(P) \cap S|).$$

### 1.2 No negative cycles

Suppose first that for paths $P$ is a path that begins at vertex 1 and $x$ is an arbitrary vertex. Then we define

$$P * x = \begin{cases} 
(P, x) & x \notin P, \\
P(1, x) & x \in P.
\end{cases}$$

Here $P(1, x)$ is the subpath of $P$ from 1 to $x$.

**Assumption:** Suppose that $P, Q$ are paths from vertex 1 to vertex $y$. Suppose that $x \notin P$ and that $\ell(Q) \leq \ell(P)$. Then $\ell(Q * x) \leq \ell(P, x)$.

Putting $P = Q$ we see that when $\ell = \ell_{\text{reg}}$ this requires $\ell(C) \geq 0$ for a cycle $C$. Here is an example where $\ell = \ell_{\text{reg}}$ and there are no negative cycles.
**Electric cars:** Suppose when we drive along edge $e_i = (x_i, y_i), i = 1, 2, \ldots, m$. Let $P_i, i = 1, 2, \ldots, n$ be a collection of paths, where $P_1 = (1)$ and $P_i$ goes from 1 to $i$.

**Lemma 1.2.** The following is a necessary and sufficient condition for $P_1, P_2, \ldots, P_n$ to be a collection of shortest paths with start vertex 1:

$$\ell(P_y) \leq \ell(P_x * y) \text{ for all } (x, y) \in E.$$  \hfill (2)

**Proof.** It is clear that (2) is necessary. If it fails then $P_x * y$ is “shorter” than $P_y$.

Suppose that (2) holds. Let $P = (1 = x_0, x_1, x_2, \ldots, x_k = i)$ be a path from 1 to $i$. We show by induction on $j$ that

$$\ell(P_{xj}) \leq \ell(P(1, x_1, x_2, \ldots, x_j)).$$  \hfill (3)

Now when $j = 0$, both sides of (3) are zero. Then if it holds for some $j \geq 0$ then (2) and the inductive assumption imply that

$$\ell(P_{xj+1}) \leq \ell(P_{xj} * x_{j+1}) \leq \ell(P(1, x_1, \ldots, x_{j+1})).$$

Thus (2) is sufficient. \hfill $\square$

**Ford’s Algorithm:**

**begin**

for $i = 2, \ldots, n$, $d(i) \leftarrow \ell(1, i)$, $P_1 \leftarrow (1, i)$;

repeat:

flag $\leftarrow 0$;

for $i = 1, 2, \ldots, m$;

begin

if $\ell(P_{yi}) > \ell(P_{xi} * y_i)$ then;

begin;

$P_{yi} \leftarrow (P_{xi} * y_i)$; $flag \leftarrow 1$;

end;

end;

until flag = 0;

end

end

**Lemma 1.3.** Ford’s algorithm terminates after at most $n$ rounds with a collection of shortest paths.

**Proof.** If the algorithm terminates then because $flag = 0$ at this point, we have that (2) holds. Thus we have shortest paths.

3
We now argue that if the minimum number of arcs in a shortest path from 1 to \( i \) has \( \nu_i \) edges then \( P_i \) is correct after \( \nu_i \) rounds. We argue by induction. This is true for \( i = 1 \) and \( \nu_1 = 0 \). Suppose that it is true for all \( i \) such that \( \nu_i \leq \nu \) and that vertex \( j \) satisfies \( \nu_j = \nu + 1 \). Let \( P = (1 = x_0, x_1, \ldots, x_{\nu+1} = j) \) be a shortest path from 1 to \( j \). Then, by induction, after \( \nu \) rounds \( P_{x_{\nu}} \) is a shortest path from 1 to \( x_{\nu} \) and then after one more round \( P_j \) is correct.

1.3 Digraphs without circuits

These are important, not least because they occur in Critical Path Analysis. Their application in this area involves computing longest paths.

1.3.1 Topological Ordering

Let the vertices of a digraph \( D = ([n], E) \) be ordered \( v_1, v_2, \ldots, v_n \). This ordering is topological if \( (v_i, v_j) \in E \) implies that \( i < j \).

**Lemma 1.4.** Digraph \( D \) has a topological ordering if and only if \( D \) has no directed circuits.

**Proof.** Suppose first that \( v_1, v_2, \ldots, v_n \) is a topological ordering and that \( D \) has a directed cycle \( v_{i_1}, v_{i_2}, \ldots, v_{i_k} \). then we have \( i_1 < i_2 < \cdots < i_k < i_1 \), contradiction.

Conversely, suppose there are no directed circuits. Let \( P = (x_1, x_2, \ldots, x_k) \) be a longest path in \( D \). Then \( x_k \) is a sink i.e. there are no directed edges \( (x_k, y) \). (If \( y \in X = \{x_1, x_2, \ldots, x_{k-1}\} \) then \( D \) contains a circuit. If \( y \notin X \) then \( (P, x) \) is longer than \( P \).)

To get a topological ordering, we let \( v_n = x_k \) and inductively order the subgraph \( H \) induced by \([n] \setminus \{v_n\}\). This is a topological ordering. If \( (v_i, v_j) \in E(H) \) then \( i < j \) because \( H \) is topologically ordered. Any other edge must be of the form \( (v_i, v_n) \).

To solve the longest path problem for paths starting at \( v_1 \), we take a topological ordering and then compute \( d(v_1) = 0 \) and then for \( j \geq 2 \),

\[
    d(v_j) = \max \{d(v_i) + \ell(v_i, v_j) : i < j \text{ and } (v_i, v_j) \in E\}. \tag{4}
\]

**Lemma 1.5.** Equation (4) computes the value of a longest path from \( v_1 \) to every other vertex.

**Proof.** That \( d(v_j) \) is correct follows by induction on \( j \). It is trivially true for \( j = 0 \) and then for \( j > 0 \) we use the fact if \( P = (x_1 = v_1, x_2, \ldots, x_k = v_j) \) is a longest path from \( v_1 \) to \( v_j \) then (i) \( x_{k-1} = v_l \) for some \( l < j \) and (ii) \( (x_1, x_2, \ldots, x_{k-1}) \) is a longest path from \( v_1 \) to \( v_l \) and (iii) \( \ell(P) = d(v_l) + \ell(v_l, v_j) \).
Critical Path Analysis: Imagine that a project consists of $n$ activities.

Making a cup of tea:

1. Get a cup from the cupboard.
2. Get a tea bag.
3. Fill the kettle with water.
4. Boil the water.
5. Pour water into cup.
6. Allow to brew.

We define a digraph with $n$ vertices, one for each activity and an edge $(i, j)$ if (i) activity $j$ cannot start until activity $i$ has been completed but (ii) only include $(i, j)$ if it is not implied by a path $(i, k, j)$. Each edge $(i, j)$ has a length equal to the estimated duration of the activity $i$.

Tea Digraph:

![Tea Digraph Diagram]

Associate a time $t_i$ to start activity $i$. Then $t_i$ is the length of the longest path to vertex $i$. The estimated completion time of the project is then the length of the longest path to FINISH.

2 Assignment Problem

A matching $M$ in a graph is a set of vertex disjoint edges. A vertex $v$ is covered by $M$ if there exists $e \in M$ such that $v \in e$. A matching $M$ is perfect if every vertex of $G$ covered by $M$.

For the complete bipartite graph $K_{A,B}$ on vertex set $A = \{a_i : i \in [n]\}, B = \{b_i : i \in [n]\}$, perfect matchings can be represented by permutations of $n$ i.e $M = \{(a_i, b_{\pi(i)}) : i \in [n]\}$.

Given a cost matrix $(c(i, j))$, the cost of a perfect matching $M = M(\pi)$ be given by

$$c(M) = \sum_{i=1}^{n} c(i, \pi(i)).$$

The assignment problem is that of finding a perfect matching of minimum cost.
2.1 Alternating paths

Given a matching $M$, a path $P = (e_1, e_2, \ldots, e_k)$ (as a sequence of edges) is alternating if the edges alternate between being in $M$ and not in $M$.

An alternating path is augmenting if it begins and ends at uncovered vertices. If $P$ is augmenting with respect to matching $M$, then $M' = M \oplus P$ is also a matching and $|M'| = |M| + 1$.

2.2 Successive shortest path algorithm

The algorithm produces a sequence $M_1, M_2, \ldots, M_n$ where $M_k$ is a minimum cost matching from $[k]$ to $[k]$. It begins with $M_1 = (1,1)$.

Suppose that $k > 1$ and that we have constructed $M_{k-1} = \{(a_i, b_{\pi(i)}): i = 1, 2, \ldots, k-1\}$.

The graph $\Gamma_k$ is the complete graph $K_{A_k,B_k}$. The digraph $\vec{\Gamma}_k$ on vertex set $A_k = \{a_i: i \in [k]\}$, $B_k = \{b_i: i \in [k]\}$ is defined as follows. The directed edges are $X = \{(b_{\pi(i)}, a_i): i \in [k-1]\}$ and $Y = \{(a_i, b_j): i \in [k], j \in [k], j \neq \pi(i)\}$. The edge $(b_{\pi(i)}, a_i) \in X$ is given length $-c(i, \pi(i))$ and the edge $(i, j) \in Y$ is given length $c(i, j)$.

We observe the following:

- If $M$ is a perfect matching of $\Gamma_k$, then $M \oplus M_{k-1}$ consists of a collection $C_1, \ldots, C_p$ of vertex disjoint alternating cycles plus an augmenting path from $a_k$ to $b_k$.

- $c(M) - c(M_{k-1}) = \sum_{i=1}^{p} \ell(C_i) + \ell(P)$

  where length $\ell$ is defined with respect to $\vec{\Gamma}_k$.

- $\ell(C_i) \geq 0$ for all $i$. Otherwise $M_{k-1} \oplus C_i$ is a matching of $\Gamma_{k-1}$ with a cost $c(M_{k-1}) + \ell(C_i) < c(M_{k-1})$.

It follows from the above that to find a minimum cost matching of $\Gamma_k$, we should find a shortest path in $\vec{\Gamma}_k$ from $a_k$ to $b_k$. Second, because $\vec{\Gamma}_k$ has no negative circuits, we can apply Ford’s algorithm to find this path.

2.3 Linear Programming Solution – Hungarian Algorithm

Consider the linear program ALP:

$$\text{Minimize } \sum_{i=1}^{n} \sum_{j=1}^{n} c_{i,j}x_{i,j}$$

(5)
Subject to
\[ \sum_{j=1}^{n} x_{i,j} = 1 \quad \text{for } i = 1, 2, \ldots, n. \] (6)
\[ \sum_{i=1}^{n} x_{i,j} = 1 \quad \text{for } j = 1, 2, \ldots, n. \] (7)
\[ x_{i,j} \geq 0 \quad \text{for } i, j = 1, 2, \ldots, n. \] (8)

The assignment problem is the solution to ALP where we replace (8) by
\[ x_{i,j} = 0 \text{ or } 1 \quad \text{for } i, j = 1, 2, \ldots, n. \] (9)

This is because (6), (7) force the set \{ (i, j) : x_{i,j} = 1 \} to be a perfect matching and (5) is then the cost of this matching.

In general replacing non-negativity constraints (8) by integer constraints (9) makes an LP hard to solve. Not however in this case.

The dual of ALP is the linear program DLP:
Maximize \[ \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j \] (10)
Subject to \[ u_i + v_j \leq c(i, j) \quad \text{for } i, j = 1, 2, \ldots, n. \] (11)

The primal-dual algorithm that we describe relies on complimentary slackness to find a solution.

**Complimentary Slackness**: If a feasible solution \( \mathbf{x} \) to ALP and a feasible solution \( \mathbf{u}, \mathbf{v} \), to DLP satisfy
\[ x_{i,j} > 0 \text{ implies that } u_i + v_j = c(i, j). \] (12)
then \( \mathbf{x} \) solves ALP and \( \mathbf{u}, \mathbf{v} \), solves DLP. For then
\[ 0 = \sum_{i=1}^{n} \sum_{j=1}^{n} (c(i, j) - u_i - v_j) x_{i,j} = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{i,j} x_{i,j} - \left( \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j \right), \] (13)
and the two solutions have the same objective value.

(We have used \( \sum_{i=1}^{n} u_i \sum_{j=1}^{n} x_{i,j} = \sum_{i=1}^{n} u_i \), which follows from (6) etc.)

The steps of the Primal-Dual algorithm are as follows:

**Step 1** Choose an initial dual feasible solution. E.g. \( v_j = 0, j \in [n] \) and \( u_i = \min_j c(i, j) \).

**Step 2** Given a dual feasible solution, \( \mathbf{u}, \mathbf{v} \), define the graph \( K_{\mathbf{u}, \mathbf{v}} \) to be the bipartite graph with vertex set \( A, B \) and an edge \( (i, j) \) whenever \( u_i + v_j = c(i, j) \).

**Step 3** Find a maximum size matching \( M \) in \( K_{\mathbf{u}, \mathbf{v}} \).

**Step 4** If \( M \) is perfect then (12) holds and \( M \) provides a solution to the assignment problem.
Step 5 If $M$ is not perfect, update $u,v$ and go to Step 3.

To carry out Step 3, we proceed as follows:

**Step 3a** Begin with an arbitrary matching $M$ of $K_{u,v}$.

**Step 3b** Let $A_U$ denote the set of vertices in $A$ not covered by $M$.

**Step 3c** Let $\overrightarrow{K}_{u,v}$ be the digraph obtained from $K_{u,v}$ by orienting matching edges from $B$ to $A$ and other edges from $A$ to $B$.

**Step 3d** Let $A_M, B_M$ denote the set of vertices in $A, B$ that are reachable by a path in $\overrightarrow{K}_{u,v}$ from $A_U$. Such paths are necessarily alternating.

**Step 3e** If there is a vertex $b \in B_M$ that is not covered by $M$ then there is an augmenting path $P$ from some $a \in A_U$ to $v$. In this case we use $P$ to construct a matching $M'$ with $|M'| > |M|$. We then go to Step 3b, with $M$ replaced by $M'$. Otherwise, Step 3 is finished.

To carry out Step 5, we assume that we have finished Step 3 with $M, A_M, B_M$. We then let

$$\theta = \min \{ c_{i,j} - u_i - v_j : a_i \in A_M, b_j \notin B_M \} > 0.$$ We know that $\theta > 0$. Otherwise, if $a_i, b_j$ is the minimising pair, then we should have put $b_j \in B_M$.

We then amend $u,v$ to $u^*, v^*$ via

$$u_i^* = \begin{cases} u_i + \theta & a_i \in A_M, \\ u_i & \text{Otherwise.} \end{cases} \quad \text{and} \quad v_j^* = \begin{cases} v_j - \theta & j \in B_M, \\ v_j & \text{Otherwise.} \end{cases}$$

Observe the following:

1. $u^*, v^*$ is feasible for DLP. $u_i^* + v_j^* \leq u_i + v_j$ except for the case where $a_i \in A_M, b_j \notin B_M$ and $\theta$ is chosen so that the increase maintains feasibility.

2. If $b \in B_M$ for the pair $u,v$ then it will stay in $B_M$ when we replace $u,v$ by $u^*, v^*$. This is because there is a path $P = (a_{i_1} \in A_U, b_i, \ldots, a_{i_k}, b_{i_k} = b)$ such that each edge of $P$ contains one vertex in $A_M$ and one vertex in $B_M$. Hence the sum $u_i + v_j$ is unchanged for edges along $P$.

3. A vertex $b \notin B_M$ contained in a pair that defines $\theta$ will be in $B_M$ when we replace $u,v$ by $u^*, v^*$.

In summary: if we reach Step 4 with a perfect matching then we have solved ALP. After at most $n$ changes of $u,v$ in Step 5, the size of $M$ increases by at least one. This is because updating $u,v$ increases $B_M$ by at least one. Thus the algorithm finishes in $O(n^4)$ time. ($O(n^3)$ time if done carefully.)
### 3 Matroids and the Greedy Algorithm

Given a ground set $X$, an independence system on $X$ is collection of subsets $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$ such that

\[
I \in \mathcal{I} \text{ and } J \subseteq I \text{ implies that } J \in \mathcal{I}.
\]  

(14)

**Examples**

**Ex. 1** The set $\mathcal{M}$ of matchings of a graph $G = (V, X)$.

**Ex. 2** The set of (edge-sets of) forests of a graph $G = (V, X)$.

**Ex. 3** The set of stable sets of a graph $G = (X, E)$. We say that $S$ is stable if it contains no edges.

**Ex. 4** The set of solutions to the \{0, 1\}-knapsack problem. Here we are given positive integers $w_1, w_2, \ldots, w_n, W$ and $X = [n]$ and $\mathcal{I} = \{S \subseteq [n] : \sum_{i \in S} w_i \leq W\}$.

**Ex. 5** Let $c_1, c_2, \ldots, c_n$ be the columns of an $m \times n$ matrix $A$. Then $X = [n]$ and $\mathcal{I} = \{S \subseteq [n] : \{c_i, i \in S\} \text{ are linearly independent}\}$.

An independence system is a matroid if whenever $I, J \in \mathcal{I}$ with $|J| = |I| + 1$ there exists $e \in J \setminus I$ such that $I \cup \{e\} \in \mathcal{I}$. Only Ex. 2 and 5 above are matroids. To check Ex. 5, let $A_I$ be the $m \times |I|$ sub-matrix of $A$ consisting of the columns in $I$. If there is no $e \in J \setminus I$ such that $I \cup \{e\} \in \mathcal{I}$ then $A_J = A_I M$ for some $|I| \times |J|$ matrix. But then

\[
|J| = \text{rank}(A_J) \leq \min \{\text{rank}(A_I), \text{rank}(M)\} \leq |I|,
\]

contradiction.

To check Ex. 2 we can argue (exercise) that $I \subseteq E$ defines a forest if and only if the columns corresponding to $I$ in the vertex-edge incidence matrix $M_G$ are linearly independent. ($M_G$ has a row for each vertex of $G$ and a column for each edge of $G$. The column $c_e, e = \{x, y\}$ has a one in row $x$ and a -1 in row $y$ and a zero in all other rows. It doesn’t matter which of the two endpoints is viewed as $x$.)

#### 3.1 Greedy Algorithm

Suppose that each $e \in E$ is given a weight $w_e$ and that the weight $w(I)$ of an independent set $I$ is given by $w(I) = \sum_{e \in I} c_e$. The problem we discuss is

Maximize $w(I)$ subject to $I \in \mathcal{I}$.  

9
Greedy Algorithm:
begin
    Sort $E = \{e_1, e_2, \ldots, e_m\}$ so that $w(e_i) \geq w(e_{i+1})$ for $1 \leq i < m$;
    $S \leftarrow \emptyset$;
    for $i = 1, 2, \ldots, m$;
    begin
        if $S \cup \{e_i\} \in \mathcal{I}$ then;
        begin
            $S \leftarrow S \cup \{e_i\}$;
        end;
    end;
end

Theorem 3.1. The greedy algorithm finds a maximum weight independent set for all choices of $w$ if and only if it is a matroid.

Proof. Suppose first that the Greedy Algorithm always finds a maximum weight independent set. Suppose that $\emptyset \neq I, J \in \mathcal{I}$ with $|J| = |I| + 1$. Define

$$w(e) = \begin{cases} 
1 + \frac{1}{2^{|I|}} & e \in I, \\
1 & e \in J \setminus I, \\
0 & e \notin I \cup J.
\end{cases}$$

If there does not exist $e \in J \setminus I$ such that $I \cup \{e\} \in \mathcal{I}$ then the Greedy Algorithm will choose the elements of $I$ and stop. But $I$ does not have maximum weight. Its weight is $|I| + 1/2 < |J|$. So if Greedy succeeds, then (14) holds.

Conversely, suppose that our independence system is a matroid. We can assume that $w(e) > 0$ for all $e \in E$. Otherwise we can restrict ourselves to the matroid defined by $\mathcal{I}' = \{I \subseteq E^+\}$ where $E^+ = \{e \in E : w(e) > 0\}$.

Suppose now that Greedy chooses $I_G = e_{i_1}, e_{i_2}, \ldots, e_{i_k}$ where $i_t < i_{t+1}$ for $1 \leq t < k$. Let $I = e_{j_1}, e_{j_2}, \ldots, e_{j_\ell}$ be any other independent set and assume that $j_t < j_{t+1}$ for $1 \leq t < \ell$. We can assume that $\ell \geq k$, for otherwise we can add something from $I_G$ to $I$ to give it larger weight. We show next that $k = \ell$ and that $i_t \leq j_t$ for $1 \leq t \leq k$. This implies that $w(I_G) \geq w(I)$.

Suppose then that there exists $t$ such that $i_t > j_t$ and let $t$ be as small as possible for this to be true. Now consider $I = \{e_{s} : s = 1, 2, \ldots, t - 1\}$ and $J = \{e_{s} : s = 1, 2, \ldots, t\}$. Now there exists $e_{j_s} \in J \setminus I$ such that $I \cup \{e_{j_s}\} \in \mathcal{I}$. But $j_s \leq j_t \leq i_t$ and Greedy should have chosen $e_{j_s}$ before choosing $e_{i_{t+1}}$. Also, $i_k \leq j_k$ implies that $k = \ell$. Otherwise Greedy can find another element from $I \setminus I_G$ to add.