# 2 Paths

*2.1 BFS • 2.2 Dijkstra's Algorithm • 2.3 LP Formulation • 2.4 Ford's Algorithm*

SHORTEST PATH PROBLEM: Given a weighted directed graph $G$ and $a, b \in V(G)$, find a path from $a$ to $b$ of minimum weight, if one exists.

## 2.1 Breadth-First Search

Below we will go through succesively better algorithms, but we won't prove correctness for all of them, only for the last one.

**Unweighted graphs**
Note first that the greedy approach does badly for shortest path: building a path by greedily adding shortest available edge will probably not even give a path from $a$ to $b$, and if you're lucky and it does, it still need not be minimal. Fortunately, in the last lecture we saw that a BFS-tree gives shortest paths to its root. We will start from that idea, but phrase it in terms of distance sets: $D_k$ will be the set of vertices at distance $k$ from $a$.

*Notation:* In a directed graph, we define the *neighborhoods* of $S \subset V(G)$ by

$$N^{\text{out}}(S) = \{v \in V(G) \backslash S : \exists u \in S \text{ with } uv \in E(G)\},$$

$$N^{\text{in}}(S) = \{v \in V(G) \backslash S : \exists u \in S \text{ with } vu \in E(G)\}.$$

**BFS Algorithm for distance sets from $a$ for unweighted graphs**

1. Set $D_0 = \{a\}$, $k = 1$.
2. Compute $D_k = N^{\text{out}}(D_{k-1}) \backslash \left(\cup_{i=0}^{k-1} D_k\right)$.
3. If $D_k = \emptyset$, go back to 4; else set $k := k + 1$ and go back to 2;
4. Return the sets $D_i$ for $i = 0, 1, \ldots, k-1$.

*Distance from $a$ to $b$:* We have $\text{dist}(a, b) = i$ iff $b \in D_i$. More generally, if we define $d : V(G) \to \mathbb{Z}_{\geq 0}$ by $d(v) = i$ iff $v \in D_i$, then $d(v) = \text{dist}(a, v)$ for any $v \in V(G)$.
Of course, if you only cared about $b$, you could let the algorithm stop as soon as $b \in D_k$.

*Shortest $ab$-path:* We can find the shortest path from $a$ to $b$ by going backwards from $b$. If $b \in D_m$, set $v_m = b$, and repeatedly choose

$$v_i \in N^{\text{in}}(v_{i+1}) \cap D_i,$$

from $i = m - 1$ to $i = 0$. Then $v_0 = a$ and $P = av_1v_2 \cdots v_{m-1}b$ is a shortest $ab$-path.
If you want all shortest $ab$-paths, go through all possible such sequences of choices of $v_i$.

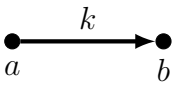
**Nonnegative weights**

Can we do the same with arbitrary weights? Think of an unweighted graph as having weight 1 at every edge. Then we could handle any graph with nonnegative integral weights by splitting an edge with weight $k$ into $k$ edges of weight 1. That gives an unweighted graph with corresponding shortest paths.

Doing this more directly gives the following not-so-efficient algorithm:

Set $D_0 = \{a\}$, and iteratively compute $D_k = \{v : \exists u \in D_{k-m}$ such that $w(uv) = m\}$ until $\bigcup D_i = V(G)$.

Clearly this is not very efficient, since every time we compute $D_k$ we have to look through all the $D_i$ we have so far. But what is worse is that many $D_k$ will be empty, and many steps will be pointless. In fact, for the graph




the algorithm will go through $k$ steps, for arbitrarily large $k$. That implies that the running time is not $O(f(n))$ for any function $f(n)$ of $n = |V(G)|$, so it certainly isn't polynomial. Fortunately, this is not too hard to fix, by skipping the empty $D_k$ as follows.

**BFS Algorithm for distance sets from $a$ for nonnegative weights**

1. Set $D_0 = \{a\}$, $d(a) = 0$, $S = D_0$;
2. Compute $m = \min(d(u) + w(uv))$ for $uv \in \delta^{\text{out}}(S)$; if $\delta^{\text{out}}(S) = \emptyset$, go to 5;
3. Compute $D_m = \{v : \exists uv \in \delta^{\text{out}}(S)$ with $d(u) + w(uv) = m\}$;
4. Add $D_m$ to $S$, set $d(v) = m$ for $v \in D_m$, go to 2;
5. Return the sets $D_i$.

*Correctness:* We won't prove correctness here, because it will follow from correctness of the more general algorithm that we see later. But it should be plausible: a vertex is only added to $D_m$ if we have a path of length $m$ to it, and there is no shorter path because then it would have been added to an earlier $D_i$.

*Polynomial running time:* In each step 2 and 3, we do $2|\delta^{\text{out}}(S)| \leq 2|E(G)|$ computations, and we will pass through at most $|V(G)|$ loops, so the algorithm is polynomial.

*Nonintegral weights:* Unlike for the previous attempt, this will work fine for nonintegral weights.

*Shortest ab-path:* We can still determine a shortest $ab$-path as before: If $b \in D_s$, find some $u \in D_r$ such that $r + w(ub) = s$, and repeat this. This is not very efficient, one could do much better by storing a "predecessor" for every vertex during the algorithm, but the idea is the same.

## 2.2 Dijkstra's Algorithm

Dijkstra's algorithm is essentially the algorithm above, but made a bit more efficient, and described more concisely. It gets rid of the $D_k$s and only uses the distance function $d(v)$. It also computes temporary estimates $d(v)$ for $v$ that are not yet in $S$, so that $d(u) + w(uv)$ does not have to be computed over and over. It does this in such a way that always $d(v) \geq \text{dist}(a, v)$, and we put $v$ into $S$ once we have $d(v) = \text{dist}(a, v)$.

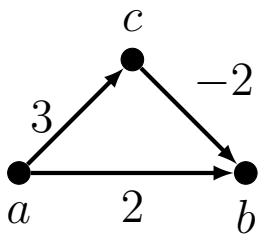
**Dijkstra's Algorithm for shortest paths from $a$ for nonnegative weights**

1. Set $d(a) = 0$ and $d(v) = \infty$ for $v \neq a$, and $S = \emptyset$;
2. Take $u \notin S$ with minimal $d(u)$ and add $u$ to $S$; if $S = V(G)$, go to 4;
3. Recompute $d(v)$ for all $v \in N^{\text{out}}(u) \backslash S$:
   if $d(v) > d(u) + w(uv)$, set $d(v) := d(u) + w(uv)$ and $p(v) = u$;
   go back to 2;
4. Return $d$ and $p$.

*Shortest ab-path:* The predecessor function $p$ defines a path: Set $v_1 = b$ and iteratively set $v_k = p(v_{k-1})$, until some $v_m = a$. then $P = av_{m-1}v_{m-2} \cdots v_2 b$ is a shortest path from $a$ to $b$.

*Variations:* Again, you could modify the algorithm if you're only interested in $b$: let it stop as soon as $b$ goes into $S$.
If you wanted not just $a$ but every shortest path, then you would have to let $p$ take sets as values and set $p(v) = \{u\}$ whenever $d(v) > d(u) + w(uv)$, and add $u$ to $p(v)$ whenever $d(v) = d(u) + w(uv)$ (i.e. whenever you find a path that is the same length as the current shortest path).

*Negative weights:* BFS and Dijkstra may not work when the graph has negative weights. For instance, for




the algorithm will end with $d(b) = 2$, when it should be 1. It will choose $b$ before $c$, set $d(b) = 2$ and put $b$ into $S$, after which it no longer updates $d(b)$.
In terms of BFS, the problem is that a negative edge with weight $w$ goes back from $D_k$ to $D_{k-w}$, so it's hard to be sure when you're done with $D_k$.
There are various things you could try to fix this, but it is probably more instructive to change perspectives, and look at this as a linear programming problem.

## 2.3 Linear Programming Formulation

*Notation:* Write $E = E(G)$ and $V = V(G)$. We will use a vector of variables $x = (x_e)_{e \in E}$, which, if all $x_e \in \{0, 1\}$, corresponds to the subgraph $H$ of $G$ with $V(H) = V(G)$ and $E(H) = \{e \in E : x_e = 1\}$.
For convenience we will also write $w_e$ for $w(e)$; we allow negative $w(e)$, but we will see below that the program is not always feasible.

**IP for shortest $ab$-path**

$$\text{minimize } \sum w_e x_e \quad \text{with } x \geq 0, \ \boxed{x \in \mathbb{Z}^{|E|}},$$

$$\sum_{e \in \delta^{\text{in}}(v)} x_e - \sum_{e \in \delta^{\text{out}}(v)} x_e = 0 \quad \text{for } v \in V \backslash \{a, b\},$$

$$\sum_{e \in \delta^{\text{in}}(b)} x_e = 1, \qquad - \sum_{e \in \delta^{\text{out}}(a)} x_e = -1.$$

Note that a feasible solution to this program is not necessarily a path, but can be a union of a path and several cycles. Still, a minimal solution to the integral program will be a path, unless there are cycles with negative weight (see below), in which case the program is unbounded (you could add that negative cycle arbitrarily many times).
For the LP relaxation of this program, the following theorem says that the same is true, with the added complication that if there are multiple shortest paths, some combinations of those would be optimal as well. In other words, the shortest paths are vertices of the corresponding polytope, and any point on the face spanned by them is also optimal.

**Theorem 2.1.** *$x$ is an optimal solution to the relaxation of the IP above if and only if $x = \sum c_i p_i$, where each $p_i$ is an integral vector and corresponds to a shortest $ab$-path $P_i$, and $\sum c_i = 1$ with $c_i \geq 0$ .*

We won't prove it here, because later in the course we will see a more general theorem that we can deduce this statement from. But a straightforward proof would not be too difficult.

**Dual Program**
To dualize the LP above, it is convenient to consider the last two constraints as special cases of the general one, which we can do if we remove all edges coming into $a$ and going out of $b$, and define the vector $f = (f_v)_{v \in V}$ by $f_b = 1$, $f_a = -1$, and all other $f_v = 0$.
Then we can take linear combinations of the constraints by a vector $(y_v)_{v \in V}$:

$$y_b - y_a = \sum_{v \in V} y_v \cdot f_v = \sum_{v \in V} y_v \cdot \left( \sum_{e \in \delta^{\text{in}}(v)} x_e - \sum_{e \in \delta^{\text{out}}(v)} x_e \right) = \sum_{uv \in E} x_{uv}(y_v - y_u),$$

and observe that if all $y_v - y_u \leq w_e$, then we have $y_b - y_a \leq \sum w_e x_e$. That gives us the dual program:

**Dual of relaxation**

$$\text{maximize } y_b - y_a \quad \text{with } y \in \mathbb{R}^{|V|},$$

$$y_v - y_u \leq w_{uv} \quad \text{for } uv \in E.$$

*Potentials:* A feasible dual solution is called a *potential* for the weight function $w$; in other words, a potential is a function $d : V(G) \to \mathbb{R}_{\geq 0}$ such that $d(v) - d(u) \leq w(uv)$ for every directed edge $uv \in E(G)$. That should look familiar: it is in fact the triangle inequality, and the distance functions $d$ as found in the algorithms above satisfy it. But not every potential gives such a distance function, since the distance functions have equality, $d(v)-d(u) = w(uv)$, along every shortest path from $a$.
Note that feasibility does not depend on $a$ or $b$, but optimality does.

*Negative cycles:* Seeing the dual program, we can deduce a condition for when shortest paths exist, or equivalently, when the dual has a feasible solution.
Given a weighted directed graph, define a *negative cycle* to be a directed cycle $C$ such that $\sum_{e \in C} w_e < 0$.

**Lemma 2.2.** *Assume $G$ is a weighted graph with the property that every vertex can be reached by a directed path from $a$.*
*Then the dual program above has a feasible solution if and only if the weighted graph has no negative cycle.*

*Proof.* If there is a negative cycle $C$, so $\sum_{e \in C} w_e < 0$, then for any potential $y$ we would have

$$0 = \sum_{uv \in C} (y_v - y_u) \leq \sum_{e \in C} w_e < 0.$$

Suppose there is no feasible solution. Let $y$ have as few edges as possible failing the condition $y_v - y_u \leq w_e$. Take any failing edge $e_1 = v_1v_2$, so $y_{v_2} - y_{v_1} > w_{e_1}$. Fix that edge by setting $y_{v_2} = y_{v_1} + w_{e_1}$. By minimality of $y$, there must be new failing edges, which must have tail $v_2$. Fix each of these: if for instance $y_{v_3} - y_{v_2} > w_{e_2}$, set $y_{v_3} = y_{v_2} + w_{e_2} = y_{v_1} + (w_{e_1} + w_{e_2})$. Again by minimality, at least one of the fixes must create a new failing edge. We can keep repeating this, so at some point we must encounter a vertex for the second time. Then we have a sequence $v_1, v_2, \ldots, v_k, \ldots, v_l$, such that $e_l = v_lv_k$ fails, i.e. $y_{v_k} - y_{v_l} > w_{e_l}$. But we have set $y_k = y_1 + \sum_{i=1}^{k-1} w_{e_i}$ and $y_l = y_1 + \sum_{i=1}^{l-1} w_{e_i}$, so we have

$$w_{e_l} < y_{v_k} - y_{v_l} = -\sum_{i=k}^{l-1} w_{e_i},$$

which means that $v_kv_{k+1} \cdots v_lv_k$ is a negative cycle.

□

Note that this means that for a graph without a negative cycle, and with all vertices reachable from $a$, the primal problem is feasible and bounded, and a shortest path exists.
If there is a negative cycle, one could still ask for the shortest path, but this turns out to be an NP-hard problem (see the Problem Set). The trick is in the fact that the LP above actually looks for minimum *walks* (like a path, but allowing vertices to be repeated, i.e. they could have cycles), and a when there is a negative cycle walks can have arbitrarily large negative weight.

## 2.4 Ford's Algorithm

**Ford's Algorithm for shortest paths from $a$ weights without negative cycles**

1. Set $d(a) = 0$ and $d(v) = \infty$ for $v \neq a$;
2. Repeat the following $|V(G)|$ times:
   - For each edge $uv \in E(G)$, do:
     - If $d(v) > d(u) + w(uv)$, then set $d(v) = d(u) + w(uv)$ and $p(v) = u$;
3. Return $d$ and $p$.

*Shortest ab-path:* The predecessor function $p$ defines a path from $a$ to $b$, as for Dijkstra's algorithm.

*Finding negative cycles:* Suppose the algorithm finishes, but we still have $d(v) > d(u)+w(uv)$ for some edge. Then set $v_n = v$, $v_{n-1}$

**Theorem 2.3.** *For a weighted directed graph $G$ without negative cycles, and a vertex $a$, Ford's algorithm correctly determines the distance function $d$ (i.e. $d(v) = dist(a, v)$ for all $v \in V(G)$), and $p$ determines a shortest path from any $b$.*

*Proof 1.* We claim that at the end of the algorithm, we have $d(v) - d(u) \leq w(uv)$ for all edges $uv$, and we have $d(v) - d(u) = w(uv)$ whenever $p(v) = u$. Both claims follow from the fact that there is a shortest path from $a$ to $u$ and to $v$ (since there is no negative cycle).
It follows that $y_v = d(v)$ defines a potential, i.e. a dual feasible solution to the program above. Given $b$, going back using $p$, we get a path from $a$ to $b$, i.e. a primal feasible solution $x$, that has $d(v) - d(u) = w(uv)$ along all its edges, Furthermore, these two feasible solutions have complementary slackness, i.e. we have $x_{uv} = 0$ whenever the dual constraint corresponding to $uv$ has slack ($d(v) - d(u) < w(uv)$). By the complementary slackness theorem (see the Introduction), both solutions are optimal, which means that $d(v)$ correctly gives the distance to $b$, and $x$ is a shortest $ab$-path. □

*Proof 2.* Let $d_i$ be the function $d$ after the $i$th iteration of step 2 (of the $|V(G)|$ iterations). Then

$$d_i(v) = \text{ minimum length of any } av\text{-path with } \leq k \text{ edges.}$$

We can prove this by induction, using that there are no negative cycles.
It follows that $d(v) = d_{|V(G)|}(v) = \text{dist}(a, v)$, since a shortest path certainly has fewer than $|V(G)|$ edges. □