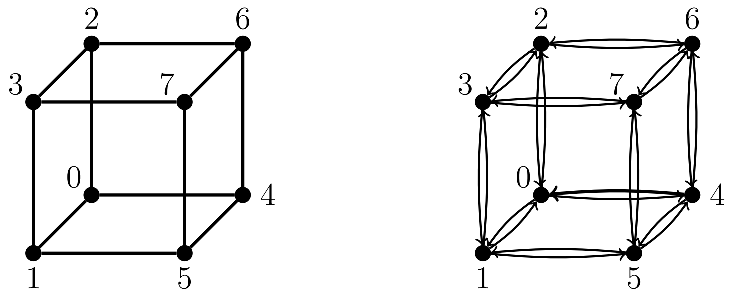
3.3.2 Routing in Hypercubes

An n-dimensional cube is called a *hypercube*. A standard definition of a hypercube is undirected, has 2^n vertices and $n2^{n-1}$ edges since every vertex has undirected edges to n other vertices. We define a directed version, where every undirected edges is replaced by two directed edges of opposite direction, thus there are $n2^n$ edges in total.

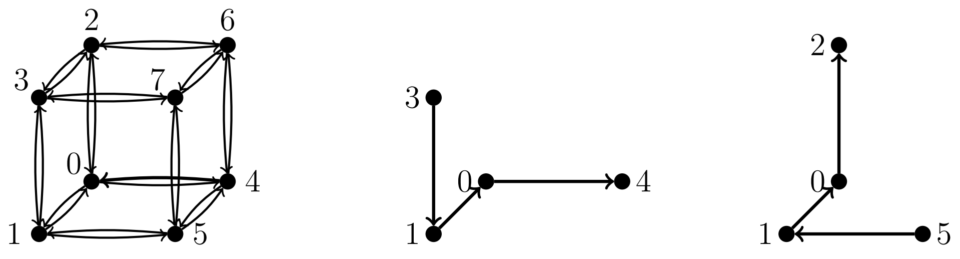


For an integer $i \in \{0, \dots, 2^n - 1\}$, let $\mathtt{bin}(i)$ be its binary representation. For two bit strings $x, y \in \{0,1\}^n$, let $h(x, y)$ be the *Hamming distance* of x and y, i.e. the number of bits in which x and y differ.

Definition 3.12. *Let $H_n = \{V_n, E_n\}$ be the directed graph defined by*

$$\begin{aligned} V_n &= \{0, \dots, 2^n - 1\} \\ E_n &= \{(i,j) \mid h(\mathtt{bin}(i), \mathtt{bin}(j)) = 1\} \end{aligned}$$

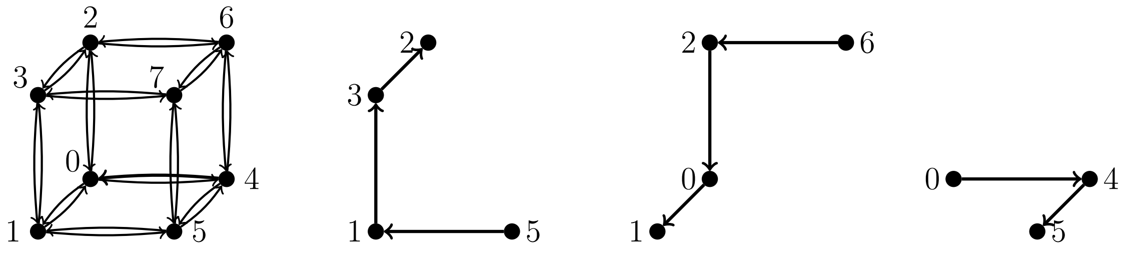
Our problem consists of simultaneously sending 2^n packages between the vertices of H_n. Each vertex $i \in V_n$ is the starting point of one package which we name a_i. The input is a permutation $\pi : V_n \to V_n$ that defines the destination $\pi(i)$ for a_i for every $i \in V_n$. The routing strategy has to choose one path $P(i)$ from i to $\pi(i)$ for every $i \in V_n$. The execution of a routing (consisting of paths $P(i)$ for all $i \in V_n$) happens in discrete time steps. At each time step, at most one package can travel over each edge. If a package travels over an edge, it arrives at the end of the edge at the end of the time step. Every package a_i follows its path $P(i)$. When a package wants to travel over an edge, then it enters a FIFO queue at the start of the edge (it two packages arive at the same time, their order is arbitrary). If the new package is the only package in the queue, then it can use the edge immediately. Otherwise, the first package in the queue travels over the edge, while the other packages in the queue wait. In the following example, we see conflicting paths for two packages. It takes four times steps until these two packages arrive.



Let $|P(i)|$ be the number of edges on $P(i)$. Then a package a_i needs at least $|P(i)|$ steps to reach $\pi(i)$, but it might need considerably more steps if it is delayed. The *execution time* of a routing is the time that it takes until all packages have arrived at their destination. We want to minimize the execution time.

We look for an *oblivious routing* strategy. That means that the path $P(i)$ may depend only on i and $\pi(i)$ and is not influenced by the destinations of other packages. Thus, the problem is to avoid that many packages choose the same path, but with a strategy that computes the path only based on start and end vertex (and possibly on random decision making).

Bit fixing paths. *Bit fixing* is a deterministic oblivious routing strategy for the hypercube. It "fixes" the bits in $\texttt{bin}(i)$ from left to right (i.e. starting at the most significant bit) such that the number i becomes $\pi(i)$. For example, if the package a_5 wants to travel from 5 to 2 in H_3, then the bit fixing path travels through the vertices with the binary representations 101, 001, 011 and 010 in this order. The bit fixing path from 6 to 1 visits the vertices with the binary representations 110, 010, 000, 001 in this order, and the path from 0 to 5 visits 000, 100 and 101.



In general, let $x_1 \dots x_n$ be the binary representation of i and let $y_1 \dots y_n$ be the binary representation of $\pi(i)$. The bit fixing path $P(i)$ has $h(\texttt{bin}(i), \texttt{bin}(\pi(i)))$ edges. Let ℓ_m for $m \in \{0, \dots, h(\texttt{bin}(i), \texttt{bin}(\pi(i)))\}$ be the index of the mth bit in which $\texttt{bin}(i)$ and $\texttt{bin}(\pi(i))$ differ. Then the mth edge on $P(i)$ goes from the vertex with binary representation $y_1 \dots y_{\ell_m - 1} x_{\ell_m} \dots x_n$ to the vertex with binary representation $y_1 \dots y_{\ell_m - 1} y_{\ell_m} x_{\ell_m + 1} \dots x_n$. Notice that this is well-defined: Only one bit changes, so two consecutive vertices on $P(i)$ are indeed always connected in H_n. Furthermore, the path $P(i)$ indeed starts in i and ends in $\pi(i)$. Finally, observe that $|P(i)| = h(\texttt{bin}(i), \texttt{bin}(\pi(i))| \leq n$ and that a bit fixing path is always a shortest path between i and $\pi(i)$. The latter holds because every edge in E_n connects two vertices with Hamming distance one, so any path between i and $\pi(i)$ has at least $h(\texttt{bin}(i), \texttt{bin}(\pi(i))$ edges.

Collisions and large worst case execution times. In our above example, the three bit fixing paths do not intersect. However, consider the following family of instances. Let n be a positive even integer. For any $i \in \{0, \dots, 2^n - 1\}$, we define $\pi(i)$ in the following way. Let $x_1 \dots x_n$ be the binary representation of i. Then $\pi(i)$ is in binary given by

$$x_{\frac{n}{2}+1} \dots x_n x_1 \dots x_{\frac{n}{2}},$$

i.e. the first and second part of the binary representation are swapped. The problem is that for all i with the same last $n/2$ bits, the bit fixing paths for this instance intersect in a vertex. Let $z_{\frac{n}{2}+1} \dots z_n$ be $n/2$ bits and let the binary representation of i be

$$x_1 \dots x_{\frac{n}{2}} z_{\frac{n}{2}+1} \dots z_n.$$

Then $\pi(i)$ has the binary representation

$$z_{\frac{n}{2}+1} \dots z_n x_1 \dots x_{\frac{n}{2}},$$

and the bit fixing path from i to $\pi(i)$ contains the vertex with the binary representation

$$z_{\frac{n}{2}+1} \dots z_n z_{\frac{n}{2}+1} \dots z_n.$$

There are $2^{n/2}$ numbers in $\{0, \dots, 2^n - 1\}$ with $z_{\frac{n}{2}+1 \dots z_n}$ as the last $n/2$ bits of their binary representation. All these numbers have the same vertex i^* in their bit fixing path. One of them is the number i^* itself, which has i^* as destination, but all $2^{n/2} - 1$ other numbers want to leave i^* again. In every time step, at most n packages can leave i^*. Thus, it takes at least $(2^{n/2} - 1)/n$ steps until all packages have left i^*, and that is a lower bound for the execution time of the whole routing strategy. Thus, the worst case execution time is $\Omega(\sqrt{2^n}/n)$ for the routing strategy that uses bit fixing paths for all packages. Surprisingly, this is true for any oblivious deterministic routing strategy.

Theorem 3.13 (Kaklamanis, Krizanc, Tsantilas [KKT91]). *For any oblivious deterministic routing strategy in H_n, there exists a permutation $\pi : V_n \to V_n$ such that the execution time of the routing strategy is $\Omega(\sqrt{2^n}/n)$ for π.*

The paper [KKT91] also contains a deterministic routing strategy that matches this worst case execution time. In light of the routing time n of a single package, we would like a faster strategy. Randomization will provide us with a way out.

Valiant's randomized routing strategy. The following strategy is attributed to Valiant and is published in [VB81, Val82]. While there exist permutations for which bit fixing paths have a lot of collisions, this is not the case if the permutation is chosen randomly (we do not prove this statement, but it is implicit in the proof that we will see). However, we do not want to force the input to be random, we want the algorithm to be randomized. Valiant's idea is to send each package a_i to $\pi(i)$ in two phases: First, the package is sent to a random intermediate location $\delta(i)$. Second, it is sent from $\delta(i)$ to $\pi(i)$. In this way, each phase has a random component: Packages are either sent to or from a random location. With high probability, bit fixing routings will be fast for both phases.

The intermediate location $\delta(i)$ is chosen uniformly at random from all V_n for all $i \in V_n$. Thus, δ is not necessarily a permutation and packages might be routed to the same intermediate location. However, the number of packages routed to the same location will with high probability not be large enough to cause high congestion in the graph. To make the two phases of the algorithm independent, we assume that all packages wait until step $4n + 1$ until they start with phase 2. We will see that with high probability, phase 1 is completed in $4n$ time steps. Phase 2 then takes at most $4n$ more time steps with high probability. The remaining part of this section consists of proving the following theorem.

Theorem 3.14. *For any $\pi : V_n \to V_n$, Valiant's randomized routing strategy has an execution time of at most $8n$ steps with probability $1 - \frac{1}{2^n}$.*

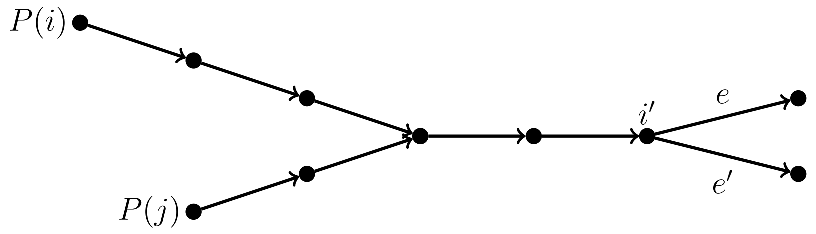
Before proving the theorem, we develop structural insights about bit fixing paths. For any $i \in V_n$ let $P(i)$ be the big fixing path from i to $\delta(i)$ during phase 1, i.e. $P(i)$ is a bit fixing path to a location chosen uniformly at random from V_n. By $|P(i)|$ we mean the number of edges on $P(i)$, and by $P(i) \cap P(j)$ for $i, j \in V_n$ we mean the set of edges that are on both paths. We define the set

$$S(i) = \{j \mid j \neq i, P(i) \cap P(j) \neq \emptyset\}$$

which is the set of all vertices j such that the bit fixing path of a_j crosses the path of a_i. The packages $a_j, j \in S(i)$ are all the packages that can possibly delay a_i when it travels to $\delta(i)$. We investigate how and how often a package $a_j, j \in S(i)$ can delay a_i. The following lemma shows that two bit fixing paths can only cross in one contiguous subpath.

Lemma 3.15. *For all $i \in V_n$ and all $j \in S(i)$, $P(i) \cap P(j)$ is a contiguous subpath of $P(i)$ and $P(j)$, respectively.*

Proof. Let i' be the last vertex of the first subpath of $P(i)$ that also lies on $P(j)$. We want to show that i' is the last vertex on $P(i)$ and $P(j)$, respectively, that lies on both paths. Let e and e' be the edges of $P(i)$ and $P(j)$, respectively, that leave i'.



Then e and e' flip different bits in the binary representation of i'. Let

$$\texttt{bin}(i') = x_1 \dots x_s \dots x_t \dots x_n$$

where x_s and x_t are the bits that are inverted, i.e. e goes to the vertex with binary representation

$$x_1 \dots \bar{x_s} \dots x_t \dots x_n$$

and e' goes to the vertex with binary representation

$$x_1 \dots x_s \dots \bar{x_t} \dots x_n$$

or vice versa. Then all future nodes on $P(i)$ and $P(j)$ differ in the sth bit, thus the paths cannot cross again. □

We learned that a package $a_j, j \in S(i)$ might delay a_i during one contiguous subpath, but once $P(i)$ and $P(j)$ diverge for the first time, a_j can no longer delay a_i. We use this fact to prove the following statement.

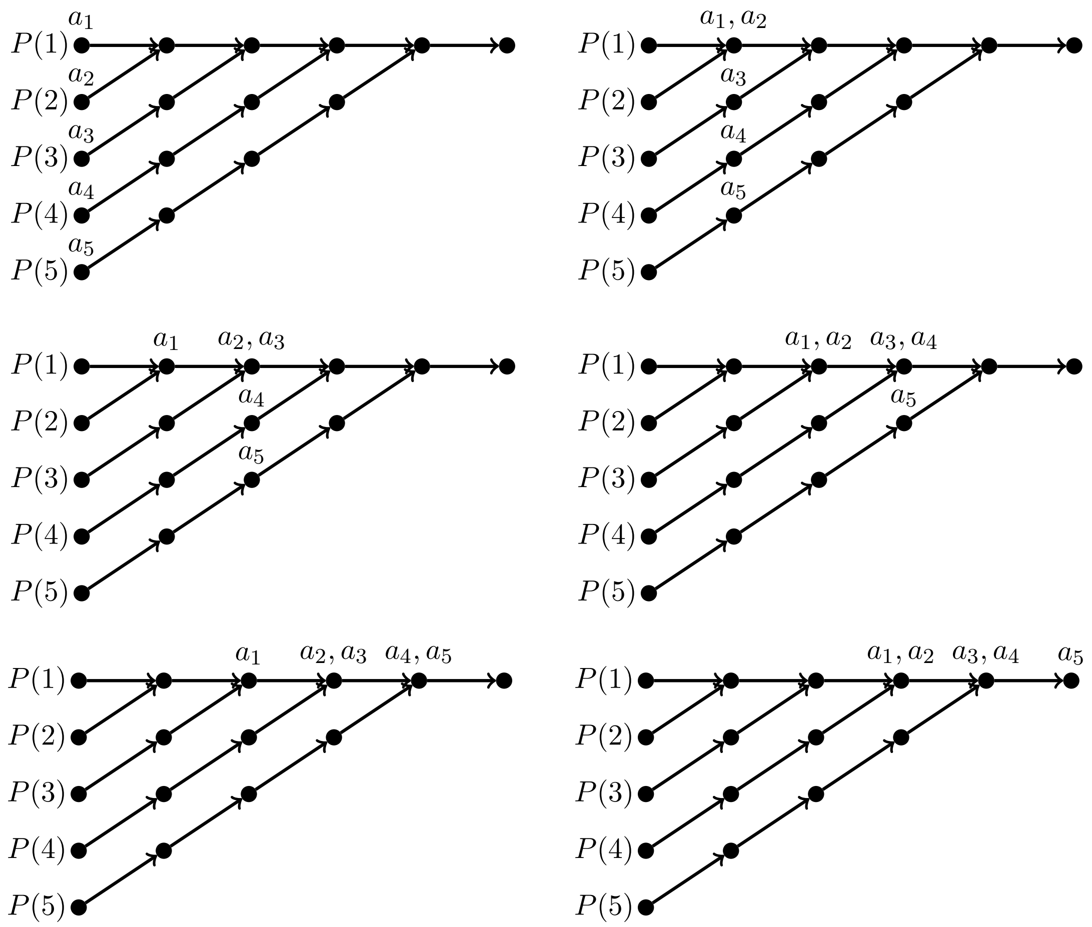
Lemma 3.16. *Let $T(i)$ be the number of steps that a_i needs to arrive at $\delta(i)$ and define the* delay *of a_i by $D(i) = T(i) - |P(i)|$. Then it holds*

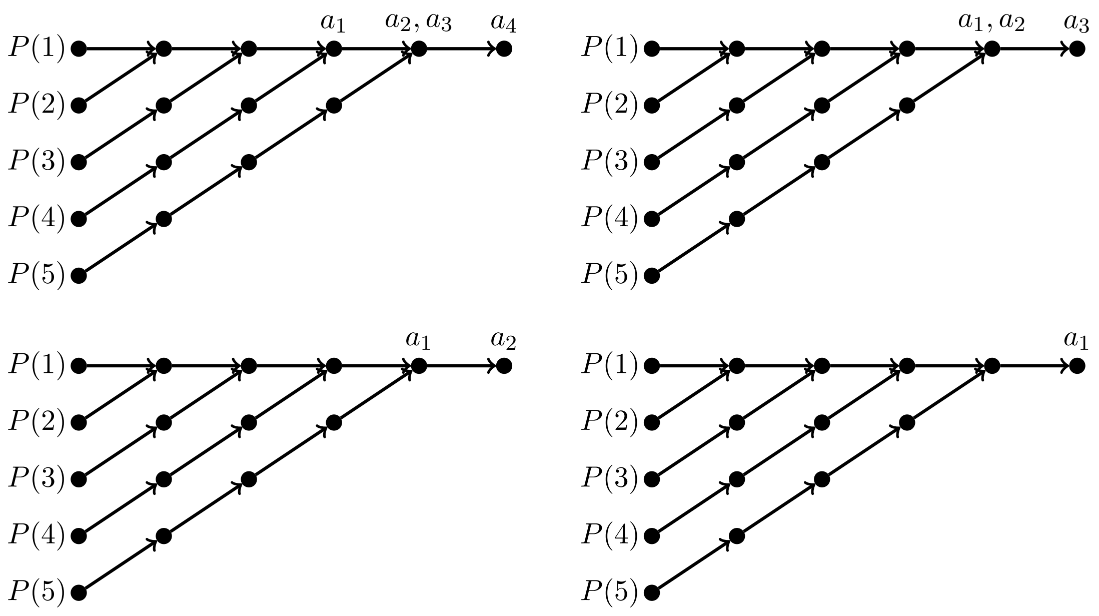
$$D(i) \leq |S(i)|$$

for all $i \in V_n$.

Proof. The package a_i waits $D(i)$ steps because it is delayed by other packages, i.e. its delay goes from 0 to $D(i)$ in steps of one. We would like to charge a distinct package in $S(i)$ for the increase from L to $L+1$ for all $L \in \{0, \ldots, D(i)\}$. The package that we charge is not necessarily the package that caused the delay for a_i. It is only important that no package in $S(i)$ gets charged twice.

In the following example, a_1 wants to travel to the rightmost vertex, the destinations of all other packages lie beyond this vertex. We assume that the FIFO queue breaks ties by sending the package with the higher index first. Even though $P(1)$ has only five edges, a_1 needs nine steps to reach its destination. It is always delayed by the same package a_2. However, the delay is still bounded by the number of crossing paths, since $P(1)$ is crossed by $P(2)$, $P(3)$, $P(4)$ and $P(5)$.





Let $P(i)$ consist of the edges $e_1, \ldots, e_{|P(i|}$ (in this order) and of the vertices $v_1, \ldots, v_{|P(i)|+1}$ (in this order). We define the *offset* of a package a_j with $j \in S(i) \cup \{i\}$ at time t as

$$\text{offset}(t, j) = t - m$$

iff a_j is at vertex v_m at time t (if a_j is not on $P(i)$ at time t, then the offset is undefined or some distinguished number like -1). Thus, the offset says whether a_j is ahead or behind a_i's schedule. In particular, offset(t, i) is the delay of a_i at the beginning of step t. The package a_i can only be delayed by packages with the same offset because only these packages are at the same vertex and potentially want to travel over the same edge as a_i. Now we define

$$\text{lucky}(t, L) = \{j \mid j \in S(i), a_j \text{ travels } e_{t+L} \text{ in step } t\}$$

for all $L \in \{0, \ldots, D(i) - 1\}$ and for all $t \in \{1 + L, \ldots, |P(i)| + L\}$. Observe that lucky$(t, L)$ either contains no or exactly one index.

Assume that the increase of a_i's delay from L to $L + 1$ happens in step t'. Thus, lucky(t', L) is nonempty because another package caused a_i to wait during step t'. Now let t'' be the largest $t \in \{1+L, \ldots, |P(i)|+L\}$ such that lucky(t, L) is nonempty. We charge the increase of $a_i's$ delay from L to $L + 1$ to the a_j with $j \in$ lucky(t'', L).

Assume that a package $a_j, j \in S(i)$ gets charged twice by this process, once for increasing the delay from L to $L + 1$ and once for increasing it from L' to $L' + 1$. By definition, $j \in$ lucky(t''_1, L) and $j \in$ lucky(t''_2, L') for $t''_1, t''_2 \in \{1 + L, \ldots, |P(i)| + L\}$. Without loss of generality, assume that $t''_1 < t''_2$. Since $t''_1 + 1 \leq t''_2 \leq |P(i)| + L$, lucky$(t''_1 + 1, L)$ is well-defined and it is empty by definition of t''_1. Observe that j has not reached the end of $P(i)$ at time t''_1 because it was still on $P(i)$ at time t''_2. However, it did not travel on the next edge, since otherwise, lucky$(t''_1 + 1, L)$ would not be empty. It cannot have been blocked by another package either, because this package then would be in lucky$(t''_1 + 1, L)$ (notice that packages that are not in $S(i)$

will never travel edges of $P(i)$ and can thus not block a_j from advancing on $P(i)$). Thus, a_j did not want to travel the next edge. It either arrived at its destination at time t_1'' or its path deviated from $P(i)$. If it arrived at its destination, then j cannot be in lucky(t_2'', L') because a_j stopped moving at time t_1''. If $P(j)$ deviated from $P(i)$, then by Lemma 3.15, $P(j)$ never joins $P(i)$ again. Thus, j cannot be in lucky(t_2'', L') in this case either. We conclude that a_j cannot be charged twice.

Since no $j \in S(i)$ can be charged twice, we proved that $D(i) \leq |S(i)|$. $\square$