Reachability

\[ \text{Reachability} = \{ \langle G, s, t \rangle \mid \text{There is a path from } s \text{ to } t \text{ in } G, \} , \]

where \( G \) is a directed graph. If \( n \) is the number of vertices and \( m \) is the number of edges of \( G \), the Reachability can be solved using BFS, which takes \( O(n + m) \) time and \( O(n) \) space. Is there a small space algorithm for Reachability? We will first show that there is a nondeterministic algorithm that uses \( O(\log n) \) space. Consider the following program.

1. Input \( G, s, t \).
2. \( \text{Current} = s \).
3. If \( \text{Current} \) equals \( t \), then Accept.
4. Guess \( \text{next} \) from the outgoing neighbors of \( \text{Current} \).
5. Go To Step 3.

Note that if there is a path from \( s \) to \( t \) in \( G \), then some computation path of this program accepts, Otherwise, no path of the program accepts. The program stores \( \text{current}, \text{next}, s \) and \( t \) n its memory. Thus the space is \( O(\log n) \).

This shows that Reachability is in \( \text{NL} \).

A Small Space Algorithm for Reachability

We now show that Reachability is in \( \text{DSPACE}(\log^2 n) \). Let \( G = (V, E) \) be the graph and assume \( G \) has \( n \) vertices. We define a predicate \( \text{Path}(u, v, k) \).

\( \text{Path}(u, v, k) \) is true if and only if there is a path from \( u \) to \( v \) whose length is at most \( k \). We make the following crucial observation. \( \text{Path}(u, v, k) \) is true if and only if there is a vertex \( w \) such that \( \text{Path}(u, w, k/2) \) and \( \text{Path}(w, v, k/2) \) are true. Our algorithm uses this observation. Now we state the algorithm. We first note that is there is a path from \( s \) to \( t \) in \( G \), then its length is at most \( n \).

1. Input: \( \langle G, s, t \rangle \).
2. If \( \text{Path}(s, t, n) \) is true then accept else reject.
We now describe the subroutine *Path*.

1. Input: $u, v, k$
2. if $u = v$ or $(u, v) \in E$ then return “True”, else
3. for each $w \in \{v_0, v_1, \ldots, v_n\}$ DO
4. if both $Path(u, w, k/2)$ and $Path(w, v, k/2)$ are true, then exit loop and return “true”
5. return “false”

The correctness of the algorithm is obvious. We can implement the subroutine *Path* using only $\log^2 n$ space using stacks. Thus Reachability is in $DSPACE(\log^2 n)$. This is a huge improvement over our earlier polynomial bound. However, the above algorithm is not practical, because it uses too much time. Verify that the above algorithm takes $O(n \log n)$ on graphs with $n$ vertices.

The above algorithm is due to Walter Savitch (Yes, the same Savitch who wrote many introductory programming text books), and is known as “Savitch’s algorithm”.

It is open if there is an algorithm for reachability that runs in polynomial time and uses space $O(\log^k n)$ for some $k > 1$.

Let $L$ be a language in $NSPACE(s(n))$. Given a string $x$ of length $n$, we can construct a graph $G_x$ of size $O(2^s(n))$ and two vertices $I$ and $F$ such that $x$ is in $L$ if and only if there is a path from $I$ to $F$ in the graph $G_x$. Since Reachability on graphs of size $2^{O(s)}$ can be solved deterministically in space $O(s^2)$, we have that $NSPACE(s(n)) \subseteq DSPACE(s^2(n))$. Thus $NL \subseteq DSPACE(\log^2 n)$ and $NPSPACE = PSPACE$. These results suggest nondeterminism does not add much power to space bounded computations.

Since Reachability can be solved in linear time, we also have that $NSPACE(s(n)) \subseteq DTIME(O(2^s(n)))$. Thus $NL \subseteq P$. 