CS 341: ALGORITHMS

Trevor Brown

trevor.brown@uwaterloo.ca

DC 2338, Office hour M3-4pm
THIS TIME

• Briefly finishing Floyd-Warshall
• Intractability (hardness of problems)
  • Decision problems
  • Complexity class P (will see another class NP next time)
  • Reductions
All-Pairs Shortest Paths

Problem 6.13

All-Pairs Shortest Paths

Instance: A directed graph $G = (V, E)$, and a weight matrix $W$, where
$W[i, j]$ denotes the weight of edge $ij$, for all $i, j \in V$, $i \neq j$.

Find: For all pairs of vertices $u, v \in V$, $u \neq v$, a directed path $P$ from $u$
to $v$ such that

$$w(P) = \sum_{ij \in P} W[i, j]$$

is minimized.

We allow edges to have negative weights, but we assume there are no
negative-weight directed cycles in $G$. 
**FLOYD-WARSHALL ALGORITHM**

- Let $D_k[i,j]$ denote the minimum-weight $(i,j)$-path in which all interior nodes are in the set of nodes $\{1 \ldots k\}$.
- Base case: $D_0 = W$
- Recurrence: $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

```python
1  FloydWarshall(V[1..n], W[1..n, 1..n])
2      D0 = weight matrix W
3      D1 = new n * n matrix
4      Dlast = pointer to D0
5      Dcurr = pointer to D1
6      for k = 1..n
7          for i = 1..n
8              for j = 1..n
9                  Dcurr[i,j] = min( Dlast[i,j], Dlast[i,k] + Dlast[k,j] )
10       swap pointers Dlast and Dcurr
11  return Dcurr
```

This returns **distances**. Can reconstruct paths from this. Easy to code, hard to understand. Not covering this.
EXAMPLE

\[
D_0 = \begin{pmatrix}
0 & 3 & \infty & \infty \\
\infty & 0 & 12 & 5 \\
4 & \infty & 0 & -1 \\
2 & -4 & \infty & 0
\end{pmatrix} \quad D_1 = \begin{pmatrix}
0 & 3 & \infty & \infty \\
\infty & 0 & 12 & 5 \\
4 & \boxed{7} & 0 & -1 \\
2 & -4 & \infty & 0
\end{pmatrix}
\]

\[
D_2 = \begin{pmatrix}
0 & 3 & \boxed{15} & \boxed{8} \\
\infty & 0 & 12 & 5 \\
4 & 7 & 0 & -1 \\
2 & -4 & \boxed{8} & 0
\end{pmatrix} \quad D_3 = \begin{pmatrix}
0 & 3 & 15 & 8 \\
\boxed{16} & 0 & 12 & 5 \\
4 & 7 & 0 & -1 \\
2 & -4 & 8 & 0
\end{pmatrix}
\]

\[
D_4 = \begin{pmatrix}
\boxed{7} & 3 & 15 & 8 \\
1 & 0 & 12 & 5 \\
\boxed{-5} & 0 & -1 \\
2 & -4 & 8 & 0
\end{pmatrix}
\]
INTRACTABILITY

Studying the hardness of problems

(knowing when to stop trying for a better solution)
Decision Problems

**Decision Problem:** Given a problem instance $I$, answer a certain question “yes” or “no”.

**Problem Instance:** Input for the specified problem.

**Problem Solution:** Correct answer (“yes” or “no”) for the specified problem instance. $I$ is a **yes-instance** if the correct answer for the instance $I$ is “yes”. $I$ is a **no-instance** if the correct answer for the instance $I$ is “no”.

**Size of a problem instance:** $\text{Size}(I)$ is the number of bits required to specify (or encode) the instance $I$. 
The Complexity Class $\mathbf{P}$

Algorithm Solving a Decision Problem: An algorithm $A$ is said to solve a decision problem $\Pi$ provided that $A$ finds the correct answer ("yes" or "no") for every instance $I$ of $\Pi$ in finite time.

Polynomial-time Algorithm: An algorithm $A$ for a decision problem $\Pi$ is said to be a polynomial-time algorithm provided that the complexity of $A$ is $O(n^k)$, where $k$ is a positive integer and $n = \text{Size}(I)$.

The Complexity Class $\mathbf{P}$ denotes the set of all decision problems that have polynomial-time algorithms solving them. We write $\Pi \in \mathbf{P}$ if the decision problem $\Pi$ is in the complexity class $\mathbf{P}$. 
Cycles in Graphs

Problem 7.1

Cycle
Instance: An undirected graph $G = (V, E)$.
Question: Does $G$ contain a cycle?

Problem 7.2

Hamiltonian Cycle
Instance: An undirected graph $G = (V, E)$.
Question: Does $G$ contain a hamiltonian cycle?

A hamiltonian cycle is a cycle that passes through every vertex in $V$ exactly once.
Knapsack Problems

Problem 7.3
0-1 Knapsack-Dec
Instance: a list of profits, \( P = [p_1, \ldots, p_n] \); a list of weights, \( W = [w_1, \ldots, w_n] \); a capacity, \( M \); and a target profit, \( T \).
Question: Is there an \( n \)-tuple \([x_1, x_2, \ldots, x_n]\) \(\in\{0, 1\}^n\) such that \(\sum w_i x_i \leq M\) and \(\sum p_i x_i \geq T\)?

Problem 7.4
Rational Knapsack-Dec
Instance: a list of profits, \( P = [p_1, \ldots, p_n] \); a list of weights, \( W = [w_1, \ldots, w_n] \); a capacity, \( M \); and a target profit, \( T \).
Question: Is there an \( n \)-tuple \([x_1, x_2, \ldots, x_n]\) \(\in\{0, 1\}^n\) such that \(\sum w_i x_i \leq M\) and \(\sum p_i x_i \geq T\)?

Relative hardness?
Polynomial-time Turing Reductions

Suppose $\Pi_1$ and $\Pi_2$ are problems (not necessarily decision problems). A (hypothetical) algorithm $B$ to solve $\Pi_2$ is called an oracle for $\Pi_2$.

Suppose that $A$ is an algorithm that solves $\Pi_1$, assuming the existence of an oracle $B$ for $\Pi_2$. ($B$ is used as a subroutine within the algorithm $A$.)

Then we say that $A$ is a Turing reduction from $\Pi_1$ to $\Pi_2$, denoted $\Pi_1 \leq^T \Pi_2$.

A Turing reduction $A$ is a polynomial-time Turing reduction if the running time of $A$ is polynomial, under the assumption that the oracle $B$ has unit cost running time.

If there is a polynomial-time Turing reduction from $\Pi_1$ to $\Pi_2$, we write $\Pi_1 \leq_T^P \Pi_2$.

Informally: Existence of a polynomial-time Turing reduction means that if we can solve $\Pi_2$ in polynomial time, then we can solve $\Pi_1$ in polynomial time.
# Travelling Salesperson Problems

### Problem 7.5

**TSP-Optimization**  
**Instance:** A graph $G$ and edge weights $w : E \rightarrow \mathbb{Z}^+$.  
**Find:** A hamiltonian cycle $H$ in $G$ such that $w(H) = \sum_{e \in H} w(e)$ is minimized.

### Problem 7.6

**TSP-Optimal Value**  
**Instance:** A graph $G$ and edge weights $w : E \rightarrow \mathbb{Z}^+$.  
**Find:** The minimum $T$ such that there exists a hamiltonian cycle $H$ in $G$ with $w(H) = T$.

### Problem 7.7

**TSP-Decision**  
**Instance:** A graph $G$, edge weights $w : E \rightarrow \mathbb{Z}^+$, and a target $T$.  
**Question:** Does there exist a hamiltonian cycle $H$ in $G$ with $w(H) \leq T$?

**Note:** TSP-Dec $\leq^T_P$ TSP-Optimal Value

**Return type** “yes/no”

**Positive edge weights**

**Return type** “a path/cycle $H$”

**Return type** “a positive integer $T$”

**Note:** TSP-Dec $\leq^T_P$ TSP-Optimization
We will use polynomial-time Turing reductions to show that different versions of the TSP are polynomially equivalent: if one of them can be solved in polynomial time, then all of them can be solved in polynomial time. (However, it is believed that none of them can be solved in polynomial time.)

- We already know
  - TSP-Dec $\leq_T^{P}$ TSP-Optimal Value
  - TSP-Dec $\leq_T^{P}$ TSP-Optimization
- We show
  - TSP-Optimal Value $\leq_T^{P}$ TSP-Dec
  - TSP-Optimization $\leq_T^{P}$ TSP-Dec
**TSP-Optimal Value $\leq_T^P$ TSP-Dec**

**Algorithm:**  
`TSP-OptimalValue-Solver(G, w)`  
`external TSP-Dec-Solver`  
`hi ← $\sum_{e \in E} w(e)$`  
`lo ← 0`  
`if not TSP-Dec-Solver(G, w, hi) then return ($\infty$)`  
`while hi > lo`  
`mid ← $\lfloor \frac{hi + lo}{2} \rfloor$`  
`if TSP-Dec-Solver(G, w, mid)`  
`then hi ← mid`  
`else lo ← mid + 1`  
`return (hi)`

This is a standard binary search technique.

Is this a **poly-time** Turing reduction?

---

**Problem 7.6**  
**TSP-Optimal Value**  
**Instance:** A graph $G$ and edge weights $w : E \rightarrow \mathbb{Z}^+$.  
**Find:** The minimum $T$ such that there exists a hamiltonian cycle $H$ in $G$ with $w(H) = T$.

**Problem 7.7**  
**TSP-Decision**  
**Instance:** A graph $G$, edge weights $w : E \rightarrow \mathbb{Z}^+$, and a target $T$.  
**Question:** Does there exist a hamiltonian cycle $H$ in $G$ with $w(H) \leq T$?
To get the process started, we need to determine an initial finite interval in which the optimal value must occur. The trivial lower bound is 0 (because all edge weights are positive) and the trivial upper bound is the sum of all the edge weights.

The complexity of the binary search is logarithmic in the length of the interval being searched, i.e., \( O(\log_2 \sum_{e \in E} w(e)) \).

The size of the problem instance is

\[
\text{Size}(I) = \text{Size}(G) + \text{Size}(w) \\
\in \Theta(\text{Size}(G) + \sum_{e \in E} \Theta(\log w(e))) \\
= \Theta(\text{Size}(G) + \sum_{e \in E} \log w(e))
\]

Observe: total binary search runtime is at most

\[
c \log \sum_{e \in E} w(e) \leq c \log \prod_{e \in E} w(e) = c \sum_{e \in E} \log w(e) \in O(\text{Size}(I))
\]
**Algorithm:** \( TSP-OptimalValue-Solver(G, w) \)

1. **external** \( TSP-Dec-Solver \)
2. \( hi \leftarrow \sum_{e \in E} w(e) \)
3. \( lo \leftarrow 0 \)
4. if not \( TSP-Dec-Solver(G, w, hi) \) then return \( (\infty) \)
5. while \( hi > lo \)
   - do \( \left\{ \right. \)
     - \( mid \leftarrow \left\lfloor \frac{hi+lo}{2} \right\rfloor \)
     - if \( TSP-Dec-Solver(G, w, mid) \) then \( hi \leftarrow mid \)
     - else \( lo \leftarrow mid + 1 \)
6. return \( (hi) \)

This is a standard binary search technique.

- **Poly-time initialization:** \( hi \leftarrow \sum_{e \in E} w(e) \)
- **Assumed O(1):** \( lo \leftarrow 0 \)
- **Entire loop is poly-time:** if not \( TSP-Dec-Solver(G, w, hi) \) then return \( (\infty) \)
- **Entire algorithm is poly-time!**
- **Implies TSP-Optimal Value is poly-time reducible to TSP-Dec**

- **If poly-time TSP-Dec-Solver exists, then we have poly-time TSP-Optimal-Value-Solver!**
- **Alg. remains poly-time if call to TSP-Dec-Solver runs in poly-time instead of O(1)**
Already know this call is poly-time reducible to TSP-Dec!

If removing edge $e$ removes every Hamiltonian cycle!

Then $e$ is part of every Hamiltonian cycle.

We keep precisely the edges that are part of every Hamiltonian cycle.

At the end, the graph is just $H$, and it consists of edges that are part of every Hamiltonian cycle. Must be one such cycle!

**TSP-Optimization**

**Instance:** A graph $G$ and edge weights $w : E \rightarrow \mathbb{Z}^+$.  
**Find:** A hamiltonian cycle $H$ in $G$ such that $w(H) = \sum_{e \in H} w(e)$ is minimized.
We are deleting each edge $e$ one after the other, checking to see if a minimum weight Hamiltonian cycle still exists.

If the answer is “no”, then we restore the edge $e$ and include $e$ in the minimum weight Hamiltonian cycle $H$.

The complexity of the “for” loop is $\Theta(m)$, which is $O(\text{Size}(I))$.

To avoid the use of the subroutine $TSP\text{-OptimalValue-Solver}$, simply plug the code for $TSP\text{-OptimalValue-Solver}$ into $TSP\text{-Optimization-Solver}$.
Proof of Correctness

Clearly $H$ contains a hamiltonian cycle of minimum weight $T^*$ at the end of the algorithm. We claim that $H$ is precisely a hamiltonian cycle.

Suppose not; then $C \cup \{e\} \subseteq H$, where $C$ is a hamiltonian cycle of weight $T^*$ and $e \in G \setminus C$. Consider the iteration when $e$ was added to $H$. Let $G'$ denote the graph $G$ at this point in time. $G'$ contains a hamiltonian cycle of weight $T^*$ but $G' \setminus \{e\}$ does not, so $e$ is included in $H$. We are assuming that

$$C \cup \{e\} \subseteq H,$$

which implies

$$C \subseteq H \setminus \{e\}.$$

Since $H \subseteq G'$, we have

$$C \subseteq H \setminus \{e\} \subseteq G' \setminus \{e\}.$$

Therefore $e$ would not have been added to $H$, which is a contradiction.