CSE202 Greedy algorithms

An induced subgraph of the collaboration graph (with Erdos number at most 2).
Made by Fan Chung Graham and Lincoln Lu in 2002.
4.4 Shortest Paths in a Graph

shortest path from Princeton CS department to Einstein’s house
4.4 Shortest Paths in a Graph

Shortest path tree in Bay area
Trees with at most 4 edges

1-edge  2-edge  3-edge  4-edge
$G$ is a tree on $n$ vertices.

$\iff$ $G$ is connected with no cycle.

$\iff$ $G$ is connected with $n-1$ edges.

$\iff$ $G$ is formed by adding a leaf to a tree of $n-1$ vertices.

$\iff$ There is a unique path between any two vertices.
**Shortest Path Problem**

**Shortest path network.**
- Directed graph $G = (V, E)$.
- Source $s$, destination $t$.
- Length $l_e = \text{length of edge } e$.

**Shortest path problem:** find shortest directed path from $s$ to $t$.

↑

cost of path = sum of edge costs in path
Dijkstra’s Algorithm

Dijkstra’s algorithm.

- Maintain a set of explored nodes $S$ for which we have determined the shortest path distance $d(u)$ from $s$ to $u$.
- Initialize $S = \{ s \}$, $d(s) = 0$.
- Repeatedly choose unexplored node $v$ which minimizes

\[ \pi(v) = \min_{e = (u,v) : u \in S} d(u) + \ell_e, \]

add $v$ to $S$, and set $d(v) = \pi(v)$.

shortest path to some $u$ in explored part, followed by a single edge $(u, v)$.
Dijkstra's Algorithm

Dijkstra’s algorithm.

- Maintain a set of explored nodes $S$ for which we have determined the shortest path distance $d(u)$ from $s$ to $u$.
- Initialize $S = \{ s \}$, $d(s) = 0$.
- Repeatedly choose unexplored node $v$ which minimizes

$$\pi(v) = \min_{e = (u,v) : u \in S} d(u) + \ell_e,$$

add $v$ to $S$, and set $d(v) = \pi(v)$.

shortest path to some $u$ in explored part, followed by a single edge $(u,v)$
Invariant. For each node \( u \in S \), \( d(u) \) is the length of the shortest \( s-u \) path.

Pf. (by induction on \( |S| \))

Base case: \( |S| = 1 \) is trivial.

Inductive hypothesis: Assume true for \( |S| = k \geq 1 \).

Let \( v \) be next node added to \( S \), and let \( u-v \) be the chosen edge.

The shortest \( s-u \) path plus \( (u, v) \) is an \( s-v \) path of length \( \pi(v) \).

Consider any \( s-v \) path \( P \). We’ll see that it’s no shorter than \( \pi(v) \).

Let \( x-y \) be the first edge in \( P \) that leaves \( S \), and let \( P' \) be the subpath to \( x \).

\( P \) is already too long as soon as it leaves \( S \).

\[
\ell(P) \geq \ell(P') + \ell(x,y) \geq d(x) + \ell(x, y) \geq \pi(y) \geq \pi(v)
\]

\( \uparrow \) nonnegative weights \( \uparrow \) inductive hypothesis \( \uparrow \) defn of \( \pi(y) \) \( \uparrow \) Dijkstra chose \( v \) instead of \( y \)
Dijkstra's Algorithm: Implementation

For each unexplored node, explicitly maintain $\pi(v) = \min_{e = (u,v) : u \in S} d(u) + \ell_e$.

- Next node to explore = node with minimum $\pi(v)$.
- When exploring $v$, for each incident edge $e = (v, w)$, update $\pi(w) = \min \{ \pi(w), \pi(v) + \ell_e \}$.

**Efficient implementation.** Maintain a priority queue of unexplored nodes, prioritized by $\pi(v)$.

<table>
<thead>
<tr>
<th>PQ Operation</th>
<th>Dijkstra</th>
<th>Array</th>
<th>Binary heap</th>
<th>d-way Heap</th>
<th>Fib heap †</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>$n$</td>
<td>$n$</td>
<td>$\log n$</td>
<td>$d \log_d n$</td>
<td>$1$</td>
</tr>
<tr>
<td>ExtractMin</td>
<td>$n$</td>
<td>$n$</td>
<td>$\log n$</td>
<td>$d \log_d n$</td>
<td>$\log n$</td>
</tr>
<tr>
<td>ChangeKey</td>
<td>$m$</td>
<td>$1$</td>
<td>$\log n$</td>
<td>$d \log_d n$</td>
<td>$1$</td>
</tr>
<tr>
<td>IsEmpty</td>
<td>$n$</td>
<td>$1$</td>
<td>$1$</td>
<td>$1$</td>
<td>$1$</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>$n^2$</td>
<td>$m \log n$</td>
<td>$m \log_{m/n} n$</td>
<td>$m + n \log n$</td>
<td></td>
</tr>
</tbody>
</table>

† Individual ops are amortized bounds
Dijkstra's Shortest Path Algorithm

Find shortest path from \( s \) to \( t \).
Dijkstra's Shortest Path Algorithm

S = \{ \}

PQ = \{ s, 2, 3, 4, 5, 6, 7, t \}
Dijkstra's Shortest Path Algorithm

\[ S = \{ \} \]
\[ PQ = \{ s, 2, 3, 4, 5, 6, 7, t \} \]
Dijkstra's Shortest Path Algorithm

S = \{ s \}
PQ = \{ 2, 3, 4, 5, 6, 7, t \}

distance label → 15

decrease key

53
Dijkstra's Shortest Path Algorithm

\[ S = \{ s \} \]
\[ PQ = \{ 2, 3, 4, 5, 6, 7, t \} \]
Dijkstra's Shortest Path Algorithm

S = \{ s, 2 \}
PQ = \{ 3, 4, 5, 6, 7, t \}
Dijkstra's Shortest Path Algorithm

$S = \{ s, 2 \}$
$PQ = \{ 3, 4, 5, 6, 7, t \}$
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2 \} \]
\[ PQ = \{ 3, 4, 5, 6, 7, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 6 \} \]
\[ PQ = \{ 3, 4, 5, 7, t \} \]
Dijkstra's Shortest Path Algorithm

S = \{ s, 2, 6 \}
PQ = \{ 3, 4, 5, 7, t \}
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 6, 7 \} \]

\[ PQ = \{ 3, 4, 5, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 6, 7 \} \]
\[ PQ = \{ 3, 4, 5, t \} \]
Dijkstra's Shortest Path Algorithm

S = \{ s, 2, 3, 6, 7 \}
PQ = \{ 4, 5, t \}
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 6, 7 \} \]
\[ PQ = \{ 4, 5, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 5, 6, 7 \} \]
\[ PQ = \{ 4, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 5, 6, 7 \} \]
\[ PQ = \{ 4, t \} \]
Dijkstra's Shortest Path Algorithm

$S = \{ s, 2, 3, 4, 5, 6, 7 \}$

$PQ = \{ t \}$
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 4, 5, 6, 7 \} \]

\[ PQ = \{ t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 4, 5, 6, 7, t \} \]
\[ PQ = \{ \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 4, 5, 6, 7, t \} \]
\[ PQ = \{ \} \]
Greed is good. Greed is right. Greed works. Greed clarifies, cuts through, and captures the essence of the evolutionary spirit.

- Gordon Gecko (Michael Douglas)
Coin Changing

**Goal.** Given currency denominations: 1, 5, 10, 25, 100, devise a method to pay amount to customer using fewest number of coins.

**Ex:** 34¢.
**Coin-Changing: Greedy Algorithm**

**Cashier’s algorithm.** At each iteration, add coin of the largest value that does not take us past the amount to be paid.

```
Sort coins denominations by value: \( c_1 < c_2 < \ldots < c_n \).

coins selected
S \leftarrow \emptyset
while \((x \neq 0)\) {
    let \( k \) be largest integer such that \( c_k \leq x \)
    if \((k = 0)\)
        return "no solution found"
    \( x \leftarrow x - c_k \)
    S \leftarrow S \cup \{k\}
} 
return S
```

**Q.** Is cashier’s algorithm optimal?
Coin-Changing: Analysis of Greedy Algorithm

**Theorem.** Greed is optimal for U.S. coinage: 1, 5, 10, 25, 100.

**Pf.** (by induction on x)
- Consider optimal way to change $c_k \leq x < c_{k+1}$: greedy takes coin $k$.
- We claim that any optimal solution must also take coin $k$.
  - if not, it needs enough coins of type $c_1, \ldots, c_{k-1}$ to add up to $x$
  - table below indicates no optimal solution can do this
- Problem reduces to coin-changing $x - c_k$ cents, which, by induction, is optimally solved by greedy algorithm.

<table>
<thead>
<tr>
<th>k</th>
<th>$c_k$</th>
<th>All optimal solutions must satisfy</th>
<th>Max value of coins 1, 2, ..., k-1 in any OPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>$P \leq 4$</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>$N \leq 1$</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>$N + D \leq 2$</td>
<td>$4 + 5 = 9$</td>
</tr>
<tr>
<td>4</td>
<td>25</td>
<td>$Q \leq 3$</td>
<td>$20 + 4 = 24$</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>no limit</td>
<td>$75 + 24 = 99$</td>
</tr>
</tbody>
</table>
**Observation.** Greedy algorithm is sub-optimal for US postal denominations: 1, 10, 21, 34, 70, 100, 350, 1225, 1500.

**Counterexample.** 140¢.
- Greedy: 100, 34, 1, 1, 1, 1, 1.
- Optimal: 70, 70.
Selecting Breakpoints
Selecting breakpoints.
- Road trip from Princeton to Palo Alto along fixed route.
- Refueling stations at certain points along the way.
- Fuel capacity = $C$.
- Goal: makes as few refueling stops as possible.

**Greedy algorithm.** Go as far as you can before refueling.
Selecting Breakpoints: Greedy Algorithm

Truck driver’s algorithm.

Sort breakpoints so that: \(0 = b_0 < b_1 < b_2 < \ldots < b_n = L\)

\[
\begin{align*}
S & \leftarrow \{0\} \quad \text{breakpoints selected} \\
x & \leftarrow 0 \quad \text{current location}
\end{align*}
\]

\[
\text{while } (x \neq b_n) \\
\quad \text{let } p \text{ be largest integer such that } b_p \leq x + C \\
\quad \text{if } (b_p = x) \\
\quad \quad \text{return } \text{"no solution"} \\
\quad \quad x \leftarrow b_p \\
\quad \quad S \leftarrow S \cup \{p\} \\
\text{return } S
\]

Implementation. \(O(n \log n)\)

- Use binary search to select each breakpoint \(p\).
**Theorem.** Greedy algorithm is optimal.

**Pf.** (by contradiction)
- Assume greedy is not optimal, and let’s see what happens.
- Let $0 = g_0 < g_1 < \ldots < g_p = L$ denote set of breakpoints chosen by greedy.
- Let $0 = f_0 < f_1 < \ldots < f_q = L$ denote set of breakpoints in an optimal solution with $f_0 = g_0$, $f_1 = g_1$, $\ldots$, $f_r = g_r$ for largest possible value of $r$.
- Note: $g_{r+1} > f_{r+1}$ by greedy choice of algorithm.

The diagram shows:
- **Greedy:** $g_0, g_1, g_2, \ldots, g_r, g_{r+1}$
- **OPT:** $f_0, f_1, f_2, \ldots, f_r, f_{r+1}, \ldots, f_q$

Why doesn’t optimal solution drive a little further?
Selecting Breakpoints: Correctness

Theorem. Greedy algorithm is optimal.

Pf. (by contradiction)

- Assume greedy is not optimal, and let's see what happens.
- Let $0 = g_0 < g_1 < \ldots < g_p = L$ denote set of breakpoints chosen by greedy.
- Let $0 = f_0 < f_1 < \ldots < f_q = L$ denote set of breakpoints in an optimal solution with $f_0 = g_0, f_1 = g_1, \ldots, f_r = g_r$ for largest possible value of $r$.
- Note: $g_{r+1} > f_{r+1}$ by greedy choice of algorithm.

Greedy:

\begin{align*}
&g_0 & g_1 & g_2 & \cdots & g_r & g_{r+1} \\
\text{OPT:} & f_0 & f_1 & f_2 & \cdots & f_r & f_{r+1}
\end{align*}

another optimal solution has one more breakpoint in common $\Rightarrow$ contradiction
The question of whether computers can think is like the question of whether submarines can swim.

Do only what only you can do.

In their capacity as a tool, computers will be but a ripple on the surface of our culture. In their capacity as intellectual challenge, they are without precedent in the cultural history of mankind.

The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence.

APL is a mistake, carried through to perfection. It is the language of the future for the programming techniques of the past: it creates a new generation of coding bums.