1-4. Reduce the satisfiability problem for the formula \((\neg a \lor b) \land (a \lor \neg b) \land (\neg a \lor \neg b \lor \neg c)\) to:

- 3-coloring, 
- clique
- subset sum problem, and
- interval computations.

\[
\begin{align*}
\text{3-coloring} & \quad (\neg a \lor b) \land (a \lor \neg b) \land (\neg a \lor \neg b \lor \neg c) \\
\end{align*}
\]
1-4. Reduce the satisfiability problem for the formula \((\neg a \lor b) \land (a \lor \neg b) \land (\neg a \lor \neg b \lor c)\) to:

- 3-coloring, clique, subset sum problem, and interval computations.

3 coloring
3-coloring
(2) \[
\begin{align*}
\text{clique} & \\
& \quad \frac{(\neg a \lor b) \land (a \lor \neg b) \land (\neg a \lor \neg b \lor \neg c)}{}
\end{align*}
\]

\text{clique size is 3.}

\begin{align*}
a &= T \\
b &= T \\
c &= F
\end{align*}
clique:

\((\overline{a} \lor b) \land (a \lor \overline{b}) \land (\overline{a} \lor \overline{b} \lor c)\)

\[\begin{align*}
a &= F \\
b &= F \\
c &= F
\end{align*}\]
Subset sum problem:

\[(a \lor b) \land \neg(a \lor b) \land \neg(a \lor \neg a \lor \neg c)\]

<table>
<thead>
<tr>
<th></th>
<th>V₁</th>
<th>V₂</th>
<th>V₃</th>
<th>C₁</th>
<th>C₂</th>
<th>C₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(\neg a)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(\neg b)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(\neg c)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

a = T  \quad b = T  \quad c = F

The SAT problem for the formula
\[(a \lor b) \land \neg(a \lor b) \land \neg(a \lor \neg a \lor \neg c)\] is reduced to the subset sum problem.
The subset sum problem is defined as:

\[(\neg a \lor b) \land (a \lor \neg b) \land (\neg a \lor \neg b \lor \neg c)\]

<table>
<thead>
<tr>
<th></th>
<th>(v_1)</th>
<th>(v_2)</th>
<th>(v_3)</th>
<th>(c_1)</th>
<th>(c_2)</th>
<th>(c_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>\neg a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>\neg a</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>\neg c</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\(a = T\)
\(b = T\)
\(c = F\)
\( (\overline{a} \lor b) \land (a \lor \overline{b}) \land (\overline{a} \lor \overline{b} \lor c) \)

**Subset Sum:**

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>c_1</th>
<th>c_2</th>
<th>c_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>f_1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>f_2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>f_3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>f_4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

a = F  
\( b = F \)  
\( c = F \)
Here $s_i, g_i$ i.e., $1, 2, 3$ are auxiliary variables and $g_1, g_2, g_3$ are clauses.

Considering, $\neg a = T$, $\neg b = T$, $\neg c = T$, then we checked the satisfiability.
Interval computation.

\[(\neg a \lor \neg b) \quad \bot \quad (a \lor \neg b) \quad \bot \quad (a \lor \neg b) \lor \neg c\]

\[a \rightarrow x_1, \quad \neg a \rightarrow 1 - x_1\]
\[b \rightarrow x_2, \quad \neg b \rightarrow 1 - x_2\]
\[c \rightarrow x_3, \quad \neg c \rightarrow 1 - x_3\]

\[(\neg a \lor \neg b) = 1 - (1 - (1 - x_1))(1 - x_2)\]
\[= 1 - (x_1)(1 - x_2)\]

\[(a \lor \neg b) = 1 - (1 - x_1)(1 - (1 - x_2))\]
\[= 1 - (1 - x_1)(1 - x_2)\]

\[(\neg a \lor \neg b) \lor \neg c = 1 - (1 - (1 - x_1))(1 - (1 - x_2))(1 - (1 - x_3))\]
\[= 1 - (x_1)(1 - x_2)(1 - x_3)\]

\[= (1 - (x_1)(1 - x_2))(1 - (1 - x_1)(1 - x_2))(1 - (x_1)(1 - x_2)(1 - x_3))\]
4) Interval computation:

\[(\neg a \lor b) \land (a \lor \neg b) \land (\neg a \lor \neg b \lor \neg c)\]

\[
\neg a = 1 - a \\
\neg a \lor b = 1 - (1 - a)(1 - b) \\
a \lor \neg b = 1 - (1 - a)(1 - b) \\
\neg a \lor \neg b \lor \neg c = 1 - (1 - a)(1 - b)(1 - c)
\]

So,

\[
(1 - a(1 - b)) \cdot (1 - (1 - a)b) \cdot (1 - abc)
\]
\[(\overline{a} \lor b) \land (a \lor \overline{b}) \land (\overline{a} \lor \overline{b} \lor c)\]

\[
a \rightarrow \overline{x_1} \quad \overline{a} \rightarrow (1-x_1)\\
b \rightarrow \overline{x_2} \quad \overline{b} \rightarrow (1-x_2)\\
c \rightarrow \overline{x_3} \quad \overline{c} \rightarrow (1-x_3)\\
\]

\[
\overline{a} \lor b = 1 - (1 - (1-x_1)) (1-x_2)\\
a \lor \overline{b} = 1 - (1-x_1) (1 - (1-x_2))\\
\overline{a} \lor \overline{b} \lor c = 1 - (1 - (1-x_1)) (1 - (1-x_2)) (1 - (1-x_3))\\
\]

\[
(1 - (1 - (1-x_1)) (1-x_2)) , (1 - (1-x_1) (1 - (1-x_2))) , (1 - (1 - (1-x_1)) (1-x_2)) (1 - (1-x_3))
\]
Internal computation:

We know that

\[ a \land b \Rightarrow 1 - (1-a) \cdot (1-b) \]

\[ a \lor b \Rightarrow a \lor (1-b) \Rightarrow 1 - (1-a) \cdot b \]

\[ \neg a \lor b \Rightarrow 1 - a \cdot (1-b) \]

\[ \neg a \lor b \lor c \Rightarrow (\neg a) \lor (1-b) \lor (1-c) \]

\[ \Rightarrow 1 - a \cdot b \cdot c \cdot \overline{e} \]

\[ F(a, b, c) = (1 - (1-a) \cdot b) \cdot (1 - a(1-b)) \cdot (1 - abc) \]

Since \( a, b, c \) are binary numbers, so the upper bound of this expression would be 1 and lower bound would be 0.

To get the upper bound,

\[ 1 - (1-a) \cdot b = 1 \]

\[ \Rightarrow (1-a) \cdot b = 0 \]

\[ \Rightarrow a = 1 \text{ or } b = 0 \]

or

\[ 1 - a(1-b) = 1 \]

\[ a = 0 \text{ or } b = 1 \]

\[ 1 - abc = 1 \]

\[ \Rightarrow abc = 0 \]

\[ \Rightarrow a = 0, \text{ or } b = 0 \text{ or } c = 0 \]

So we can write \( \overline{y} \leq 1 \Rightarrow \overline{y} = 1 \).
5. What do we gain when we prove that a problem is NP-hard? Explain one negative consequence (what we cannot do) and one positive one (what we can do).

If a problem is NP-hard, then, unless \( P = NP \), no feasible algorithm is possible for solving all particular cases of this problem. Thus, we should not waste time looking for such a general algorithm.

By definition, the fact that a problem is NP-hard means that every problem from the class NP can be reduced to it. Thus, if we have a feasible algorithm for solving some particular cases of this problem, then reduction automatically generates algorithms for solving particular cases of all NP-problems. This reduction has been helpful in many cases.
5. What do we gain when we prove that a problem is NP-hard? Explain one negative consequence (what we cannot do) and one positive one (what we can do).

First, a problem is NP-hard, then, unless P = NP (which most computer scientists believe to be impossible), no feasible algo is possible for solving all particular cases of this problem. Then we should not waste time looking for such a general algorithm.

Secondly, by definition, the fact that a problem is NP-hard, it means that every problem from the class NP can be reduced to it. Thus if we good heuristic feasible algorithm for solving some particular cases of this problem, then reduction automatically generates algorithms for solving particular case of all NP-problems. This reduction has been helpful in many cases.

---

**Negative consequence:**

There is no way to have a feasible algorithm for solving all instances of the problem.

---

**Positive one:**

We do not waste time coming up with a general feasible algorithm. And what we can do is that we are still needed (e.g. artists, pools etc.)
5. What do we gain when we prove that a problem is NP-hard? Explain one negative consequence (what we cannot do) and one positive one (what we can do).

**Positive consequence:**

We don't waste time to come up with a general feasible algorithm.

**Negative consequence:**

No way to have a feasible algorithm for solving all instances of the problem.
5. What do we gain when we prove that a problem is NP-hard? Explain one negative consequence (what we cannot do) and one positive one (what we can do).

- If a problem is NP-hard, then unless P = NP (believe to be impossible), no feasible algorithm is possible for solving all particular case of this problem. Thus we should not waste time looking for such a general algorithm.

- By definition, the fact that a problem is NP-hard means that every problem from class NP can be reduced to it. Thus if we have a good heuristic feasible algorithm for solving some particular case of this problem, then reduction automatically generates algorithm for solving particular case of all NP-problems. This reduction has been helpful in many cases.
6. Show how to compute the product of 10 numbers in parallel if we have an unlimited number of processors. How many processors do we need and how much time will this computation take? Why do we need parallel processing in the first place?

\[
\begin{align*}
\text{proc 1} & : x_1 \cdot x_2 \\
\text{proc 2} & : x_3 \cdot x_4 \\
\text{proc 3} & : x_5 \cdot x_6 \\
\text{proc 4} & : x_7 \cdot x_8 \\
\text{proc 5} & : x_9 \cdot x_{10} \\
\text{moment 1} & : \text{prod} (x_1, x_2) \\
\text{moment 2} & : \text{prod} (x_3, x_4, x_5, x_6) \\
\text{moment 3} & : \text{prod} (x_7, x_8, x_9, x_{10}) \\
\text{moment 4} & : \text{prod} (x_1, x_2, \ldots, x_8, x_9, x_{10})
\end{align*}
\]

The number of processors = 5

Total moments of time = 4

When running an algorithm takes too much time on a computer, then the natural idea is to use several computers to implement it. That is if the problem is too costly to be solved sequentially, then we use parallel processing to solve it within less time.
6. Show how to compute the product of 10 numbers in parallel if we have an unlimited number of processors. How many processors do we need and how much time will this computation take? Why do we need parallel processing in the first place?

No of processors needed : 5
moment of time : 4.

We need parallel processing with the objective of running a program in less time. It helps to solve larger problems in less time compared to sequential computing, by processing the problem parallelly.
6. Show how to compute the product of 10 numbers in parallel if we have an unlimited number of processors. How many processors do we need and how much time will this computation take? Why do we need parallel processing in the first place?

The algorithm is given below (considering 10 numbers: $x_1, x_2, x_3, x_4, \ldots, x_{10}$):

- At $t=1$: $\text{product}(x_1, x_2)$, $\text{product}(x_3, x_4)$, $\text{product}(x_5, x_6)$, $\text{product}(x_7, x_8)$, $\text{product}(x_9, x_{10})$.
- At $t=2$: $\text{product}(x_1 \ldots x_4)$, $\text{product}(x_5 \ldots x_8)$.
- At $t=3$: $\text{product}(x_1 \ldots x_8)$.
- At $t=4$: $\text{product}(x_1 \ldots x_{10})$.

So, the number of processors = 5. ($\frac{10}{2} = 5$).

Required time = 4. (Because $\log_{10} 10 = 4$).
we need the parallel processing in first place, so that the computation time would be less, that is, to increase the speed of computation. Furthermore, we solve complex and large size of problems using faster processors.
6. Show how to compute the product of 10 numbers in parallel if we have an unlimited number of processors. How many processors do we need and how much time will this computation take? Why do we need parallel processing in the first place?

- No of processors = 5
- Time = 4

Parallel processing takes less time than solving the problem sequentially, that's why we need parallel processing in the first place.
7. If we take into account communication time, how fast can you compute the product of n numbers in parallel?

In this case, the computation is limited by the speed of the communication. Assume the communication is at most as fast as the speed of light.

An algorithm that takes $T_{\text{seq}}(n)$ can be performed sequentially and $T_{\text{par}}(n)$ in parallel.

$R = c \cdot T_{\text{par}}(n)$ is the radius of the sphere where we can fit the processors that can communicate with each other.

The volume, $V = \frac{4}{3} \pi R^3 T_{\text{par}}(n)$.

The maximum number of processors of minimum size $\Delta V$ is

$$N_{\text{proc}} \leq \frac{V}{\Delta V} \leq \frac{4}{3} \pi R^3 \frac{\Delta V}{T_{\text{par}}(n)}$$

So,

$$T_{\text{seq}}(n) \leq N_{\text{proc}} \cdot T_{\text{par}}(n)$$

$$T_{\text{seq}}(n) \leq (\text{constant}) \cdot (T_{\text{par}}(n))^{4}$$

$$= T_{\text{par}}(n) \geq \text{constant} \cdot (T_{\text{seq}}(n))^{1/4}$$

So, $(O(n)^{1/4})$
7. If we take into account communication time, how fast can you compute the product of \( n \) numbers in parallel?

In computing the product of \( n \) numbers in parallel, computation is limited by the speed of communication. Assuming that the communication is at most as fast as the speed of light, we have an algorithm that takes \( T_{\text{seq}}(n) \) when performed sequentially and \( T_{\text{par}}(n) \) in parallel.

\[ R = C \cdot T_{\text{par}}(n) \]

is the radius of a sphere where we can fit processors that can communicate with each other.

The volume is
\[ V = \frac{4}{3} \pi R^3 = \frac{4}{3} \pi \left( \frac{1}{3} \left( T_{\text{par}}(n) \right)^3 \right)^3. \]

The maximum number of processors of minimum size \( \Delta \)

\[ N_{\text{proc}} \leq \frac{V}{\Delta V} = \frac{4}{3} \frac{\pi c^3}{\Delta V} \left( T_{\text{par}}(n) \right)^3. \]

\[ T_{\text{seq}}(n) \leq N_{\text{par}}(n) \cdot T_{\text{par}}(n) \leq \frac{4}{2} \frac{\pi c^3}{\Delta V} \left( T_{\text{par}}(n) \right)^3 \left( T_{\text{par}}(n) \right) \leq \text{const} \left( T_{\text{par}}(n) \right)^4. \]

\[ \Rightarrow T_{\text{par}}(n) \geq \text{const} \left( T_{\text{seq}}(n) \right)^{1/4}. \]

\[ \text{with communication time: } T_{\text{par}} \geq \text{const} n^{1/4} \text{ time.} \]

\[ \text{without communication time: } T_{\text{par}} = \log_2(n) \text{ time.} \]
8. Suppose that we have a probabilistic algorithm that gives a correct answer half of the time. How many times do we need to repeat this algorithm to make sure that the probability of a false answer does not exceed 1%? Give an example of a probabilistic algorithm. Why do we need probabilistic algorithms in the first place?

The probability that gives a correct answer \( p_{\text{correct}} = \frac{1}{2} \).

Now

\[ P_{\text{false}} = 1 - \frac{1}{2} = \frac{1}{2}. \]

and \( \epsilon = \frac{1}{2} = 10^{-1} \).

\[ P_{\text{false}} \leq \epsilon \]

\[ \Rightarrow \left( \frac{1}{2} \right)^k \leq \frac{1}{100} \]

\[ \Rightarrow 2^k \geq 100 \]

\[ \Rightarrow k \geq \frac{\ln 100}{\ln 2} \]

\[ \Rightarrow k \geq 6.64. \]

So, \( k = 7 \).

So, the answer is 7 times we need to repeat.

A probabilistic algorithm is Monte Carlo simulation. This algorithm always returns a result, but the result may not always be correct. Thus we attempt to minimize the probability of an incorrect result and using the random element \( (x_i) \), multiple runs of the algorithm reduces the probability of incorrect result.
We need probabilistic algorithm in the first place, because,

i) Many practical problems are NP-hard.

ii) This means that, unless P=NP, no feasible algorithm is possible that solves all instances of this problem.

iii) In some cases, it is possible to solve some instances.
8. Suppose that we have a probabilistic algorithm that gives a correct answer half of the time. How many times do we need to repeat this algorithm to make sure that the probability of a false answer does not exceed 1%? Give an example of a probabilistic algorithm. Why do we need probabilistic algorithms in the first place?

\[ P_0 = \frac{1}{2}, \quad \varepsilon = 1\% = \frac{1}{100} \]

\[ (\frac{1}{2})^K \leq \frac{1}{100} \]

\[ 2^K \geq 100 \]

\[ K = 7 \]

**Example:** \( F \neq F_0 \)

\[
\begin{align*}
F(\alpha_1) & \neq F_0(\alpha_1) \\
1 - P_0 & \quad P_0 \\
\neq & \quad = \\
1 - P & \quad P_0 \\
\neq &
\end{align*}
\]

Many practical problems are \textbf{NP hard}, which means (unless \( P=NP \)), no \textbf{feasible algorithm} is possible that solves all instances of this problem. In some cases, it is possible to solve some instances.

Secondly, it can \textbf{speed up feasible algorithm}, like \textbf{monte carlo simulation}. 

file:///Q:/cs5315.17/test3.html

4/15/2017
9. Let us consider the following particular case on an Ali-Baba problem: we have 4 objects with weights 100, 200, 300, and 100, and value 150, 400, 150, and 200. The overall weight is limited by 500. Show, step by step, what solution two greedy algorithms will produce for this example.

Let's consider, \( W = 500 \) and \( w_1 = 100 \), \( w_2 = 200 \), \( w_3 = 300 \), \( w_4 = 100 \), and \( p_1 = 150 \), \( p_2 = 400 \), \( p_3 = 150 \), \( p_4 = 200 \).

Weights:

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>200</td>
<td>300</td>
<td>100</td>
</tr>
</tbody>
</table>

Such that:

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
<td>400</td>
<td>150</td>
<td>200</td>
</tr>
</tbody>
</table>

**1st Algorithm:**

**Step 1:** Since \( p_2 \) is maximum, we **pick** \( w_2 \) (since \( w_2 \leq W \)).

**Step 2:** Then since \( p_4 \) is maximum, we **pick** \( w_4 \) (since \( w_4 \leq W \)).

**Step 3:** Then for \( p_1 \), we **pick** \( w_1 \) (since \( w_1 \leq W \)), but we could not pick \( w_3 \), because if we need to satisfy the following condition:

\[
200 + 100 + 100 \leq 500
\]

\[
\Rightarrow 400 \leq 500
\]

So, for \( x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1 \), are yet the maximum price,

\[
p_2 + p_4 + p_1 = 750.
\]

which satisfies the condition:

\[
w_1 x_1 + w_2 x_2 + w_4 x_4 \leq W.
\]
2nd algorithm:

Step 1: Now, we compute the unit price:

\[ U_1 = \frac{150}{100} = 1.5 \]
\[ U_2 = \frac{300}{200} = 2 \]
\[ U_3 = \frac{150}{500} = 0.3 \]
\[ U_4 = \frac{200}{100} = 2 \]

Hence, the highest unit price is \( U_2 \) and \( U_4 \).

Step 2: We pick \( W_2 \) first (since \( W_2 \leq W \)).

Step 3: Then \( W_4 \) (since \( W_4 \leq W_2 \)).

Step 4: Then \( W_1 \) (since \( W_1 \leq W \)), but we cannot pick \( W_3 \) because the following condition needs to be satisfied:

\[ W_2 + W_4 + W_1 \leq W \implies 400 \leq 500 \]

Thus, we choose

\[ x_1 = 1, \quad x_2 = 1, \quad x_3 = 0, \quad x_4 = 1 \]

such that

\[ w_1x_1 + w_2x_2 + w_4x_4 \leq W \]

So, price = \( 150 + 400 + 200 = 750 \ldots \)
9. Let us consider the following particular case on an Ali-Baba problem: we have 4 objects with weights 100, 200, 300, and 100, and value 150, 400, 150, and 200. The overall weight is limited by 500. Show, step by step, what solution two greedy algorithms will produce for this example.

\[
\begin{array}{cccccc}
\chi & 1 & 2 & 3 & 4 & W \\
\omega & 100 & 200 & 300 & 100 & 500 \\
\rho & 150 & 400 & 150 & 200 & \\
\rho / \omega & 1.5 & 2 & 0.5 & 2 & \\
\end{array}
\]

1. Greedy algorithm 1: Pick up largest value one first.
   (1.x) \(400, 200, 150\) \((\chi_2, \chi_4, \chi_1)\)
   \(\omega = \chi_1 = 1, \chi_2 = 1, \chi_3 = 0, \chi_4 = 1\).
   \(\Sigma \chi \cdot \rho = 1 \cdot 150 + 1 \cdot 400 + 0 \cdot 150 + 1 \cdot 200 = 750\)
   \(\Sigma \chi \cdot \omega = 1 \cdot 100 + 1 \cdot 200 + 0 \cdot 300 + 1 \cdot 100 = 400 \leq 500\)

2. Greedy algorithm 2: Pick up largest \(\rho / \omega\) value first.
   (1.x) \(2, 2, 1.5\) \((\chi_2, \chi_4, \chi_1)\)
   \(\chi_1 = 1, \chi_2 = 1, \chi_3 = 0, \chi_4 = 1\).
   \(\Sigma \chi \cdot \rho = 1 \cdot 150 + 1 \cdot 400 + 0 \cdot 150 + 1 \cdot 200 = 750\)
   \(\Sigma \chi \cdot \omega = 1 \cdot 100 + 1 \cdot 200 + 0 \cdot 300 + 1 \cdot 100 = 400 \leq 500\)

So, we can use both algorithm in this case.
9. Let us consider the following particular case on an Ali-Baba problem: we have 4 objects with weights 100, 200, 300, and 100, and value 150, 400, 150, and 200. The overall weight is limited by 500. Show, step by step, what solution two greedy algorithms will produce for this example.

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w_i )</td>
<td>100</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td>( p_i )</td>
<td>150</td>
<td>400</td>
<td>150</td>
</tr>
<tr>
<td>( p_i/w_i )</td>
<td>1.5</td>
<td>2</td>
<td>1.5</td>
</tr>
</tbody>
</table>

All baba prob says, find values \( x_1, x_2, x_3, x_4 \) from \( 0, 1 \) of which the following inequalities hold:

\[
\begin{align*}
    x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4 & \leq W \\
    x_1 p_1 + x_2 p_2 + x_3 p_3 + x_4 p_4 & \geq P
\end{align*}
\]

**Algorithm 1:** Pick the most expensive (greater \( p_i \)) object.

Pick \( x_2 = 1, x_4 = 1, x_1 = 1 \) and \( x_3 = 0 \)

\[
\begin{align*}
    W &= 100 + 200 + 100 = 400 & \text{holds the inequality} \\
    P &= 150 + 400 + 200 = 750
\end{align*}
\]

**Algorithm 2:** Pick the most valuable (greater \( p_i/w_i \)) object

Pick \( x_2 = 1, x_4 = 1, x_1 = 1 \) and \( x_3 = 0 \)

\[
\begin{align*}
    W &= 100 + 200 + 100 = 400 & \text{holds the inequality} \\
    P &= 150 + 400 + 200 = 750
\end{align*}
\]

4/15/2017
10. Give two examples of how non-Euclidean physics can potentially help solve NP-hard problems in polynomial time.

**Example 1:** Lobachevsky geometry allows us to fit an exponential number of processors in the sphere which would allow to compute exponential time algorithms in polynomial time.

**Example 2:** If we have a time machine, the time travel is possible. One approach would be leave a computer performing the computation and sending the result back in time.
10. Give two examples of how non-Euclidean physics can potentially help solve NP-hard problems in polynomial time.

The physics are based on Euclidean arguments, as stated that \( v \leq c \) and that \( v = \frac{4}{3} \pi r^3 \) and because information cannot travel faster, one can only use polynomial number of processors which are not enough to solve exponential time problems.

Non-Euclidean physics can potentially help solve NP-hard problems in polynomial time, i.e.,

**Scheme 1:** Let a computer run & send signal back in time when done.

**Scheme 2:** Run a random generator, check whether \( V_i \sim 1 \) s.t. given formula if not involve a time machine: \( \hat{E} \geq 3/p \).

**Scheme 3:** Use Locality space.
10. Give two examples of how non-Euclidean physics can potentially help solve NP-hard problems in polynomial time.

The physics are based on Euclidean arguments, we state that $U \leq C$ and that $V = \frac{4}{3} \pi r^3$, and because information can't travel faster, we can only use polynomial number of processors, which are not enough to solve exponential time problems.

Non-Euclidean physics can solve this for example:
- Lobachevsky geometry allows us to fit an exponential number of processors in the sphere, which would allow to compute exponential time algorithm in polynomial time.
- $U > C$: If speed can be faster than $C$, time travel is possible. One approach would be to leave a computer performing the computation and sending the result back in time. The other would be to exploit causality by having random

file://Q:/cs5315.17/test3.html
number generators try to find a solution and trying to provoke very low probability events (much lower than finding the correct solution) by traveling back in time and trying to disrupt causality.