EECS 3101

Prof. Andy Mirzaian

Greedy Algorithms

STUDY MATERIAL:

- [CLRS] chapter 16
- Lecture Notes 7, 8

TOPICS

- **Combinatorial Optimization Problems**
- **The Greedy Method**
- **Problems:**
  - Coin Change Making
  - Event Scheduling
  - Interval Point Cover
  - More Graph Optimization problems considered later

COMBINATORIAL OPTIMIZATION

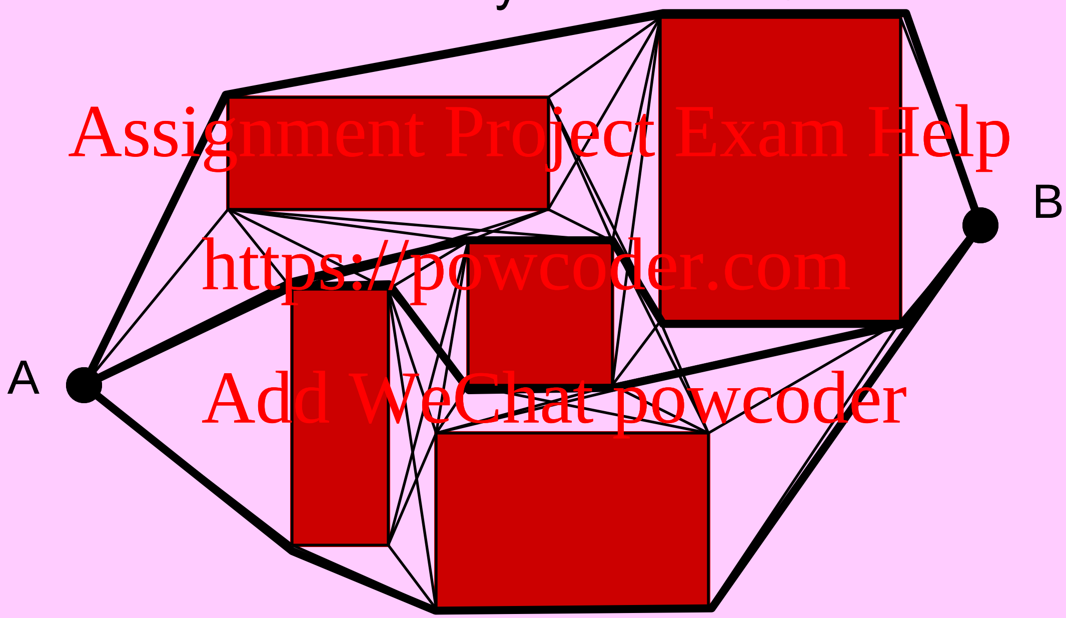
find the best solution out of finitely many possibilities.

Mr. Roboto:
Find the best path from A to B avoiding obstacles

There are infinitely many ways to get from A to B.

With brute-force there are exponentially many paths to try. ☹

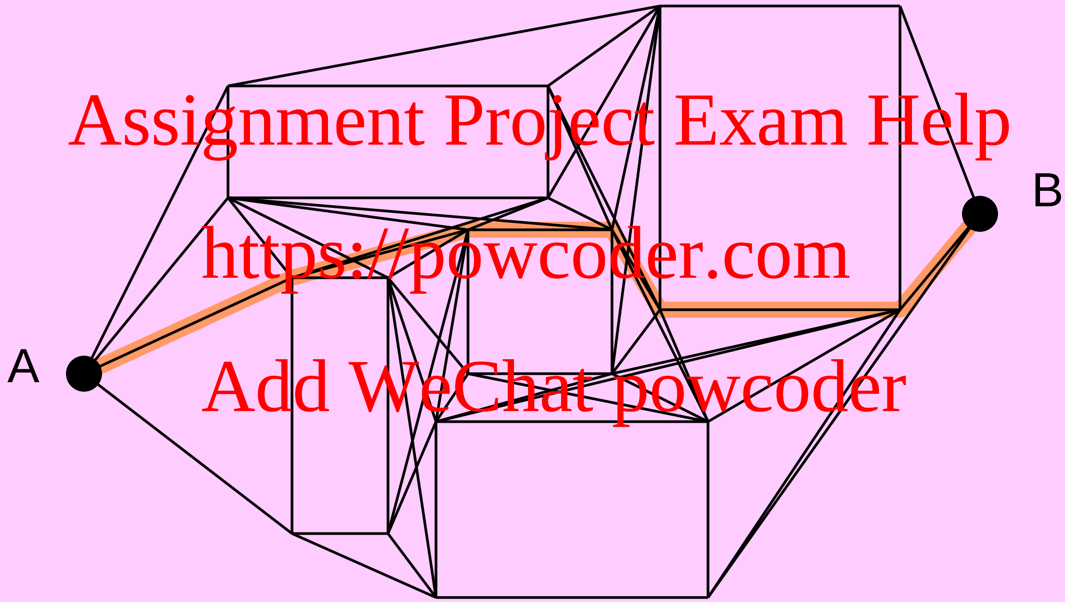
We can't try them all. ☹



There may be a simple and fast (incremental) greedy strategy. ☺

But you only need to try finitely many critical paths to find the best. ☺

Mr. Roboto:
Find the best path from A to B avoiding obstacles



The Visibility Graph: 4n + 2 vertices
(n = # rectangular obstacles)

Combinatorial Optimization Problem (COP)

INPUT: Instance I to the COP.

Feasible Set: FEAS(I) = the set of all feasible (or valid) solutions for instance I, usually expressed by a set of constraints.

Objective Cost Function:

Instance I includes a description of the objective cost function, Cost_I that maps each solution S (feasible or not) to a real number or $\pm\infty$.

Goal: Optimize (i.e., <u>minimize</u> or maximize) the objective cost function.

Optimum Set:

$\text{OPT}(I) = \{ \text{Sol} \in \text{FEAS}(I) \mid \text{Cost}_I(\text{Sol}) \le \text{Cost}_I(\text{Sol}'),\ \forall \text{Sol}' \in \text{FEAS}(I) \}$

the set of all <u>minimum</u> cost feasible solutions for instance I

Combinatorial means the problem structure implies that only a discrete & finite number of solutions need to be examined to find the optimum.

OUTPUT: A solution $\text{Sol} \in \text{OPT}(I)$, or report that $\text{FEAS}(I) = \emptyset$.

GREEDY METHOD

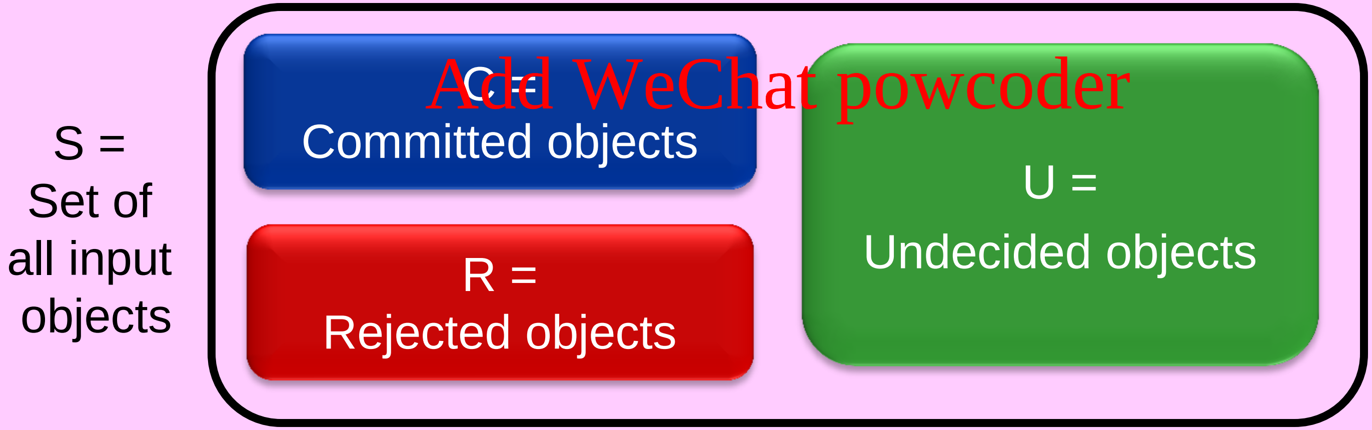
Greedy attitude:

Don't worry about the long term consequences of your current action.
Just grab what looks best right now.

- Most COPs restrict a feasible solution to be a finite set or sequence from among the objects described in the input instance.
E.g., a subset of edges of the graph that forms a valid path from A to B.

- For such problems it may be possible to build up a solution incrementally by considering one input object at a time.

- **GREEDY METHOD** is one of the most obvious & simplest such strategies:
Selects the next input object x that is **the incremental best** option at that time.
Then makes a **permanent decision** on x (without any backtracking),
either committing to it to be in the final solution, or
permanently rejecting it from any further consideration.



- Such a strategy may not work on every problem.
But if it does, it usually gives a very simple & efficient algorithm.

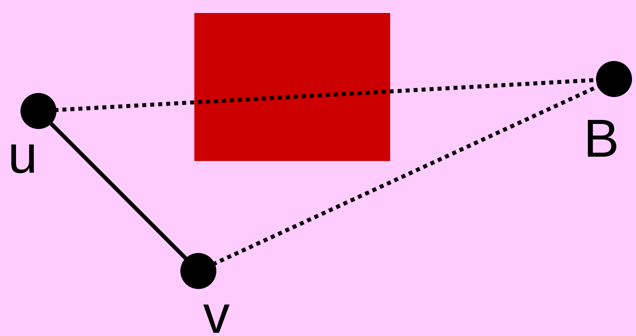
Obstacle avoiding shortest A-B path

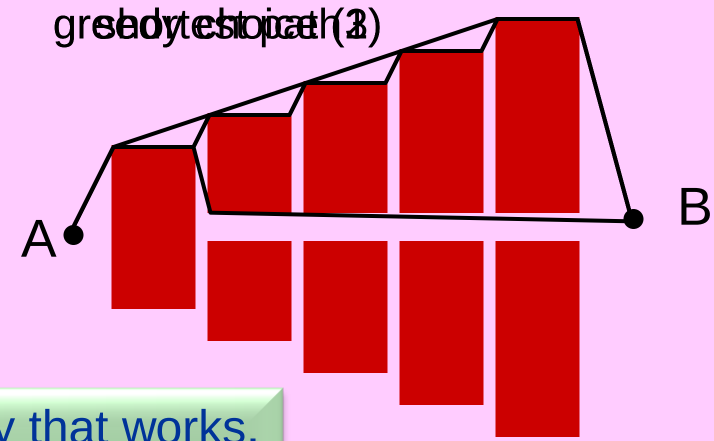
d(p,q) = straight-line distance between points p and q.
u = current vertex of the Visibility Graph we are at (u is initially A).

If (u,B) is a visibility edge, then follow that straight edge. Done.
Otherwise,
from among the visibility edges (u,v) that get you closer to B, i.e., d(v,B) < d(u,B),
make the following greedy choice for v:

(1) Minimize d(u,v) ⟵ take shortest step while making progress
(2) Minimize d(v,B) ⟵ get as close to B as possible
(3) Minimize d(u,v) + d(v,B) ⟵ stay nearly as straight as possible

Which of these greedy choices is guaranteed to produce the shortest A-B path?





Later we will investigate a greedy strategy that works.

Conflict free object subsets

S = the set of all input objects in instance $\mathcal{I}$

FEAS($\mathcal{I}$) = the set of all feasible solutions

Definition: A subset $C \subseteq S$ is "conflict free" if it is contained in some feasible solution:

$$\text{ConflictFree}_{\mathcal{I}}(C) = \begin{cases} \text{true} & \text{if } \exists Sol \in \text{FEAS}(\mathcal{I}) : C \subseteq Sol \\ \text{false} & \text{otherwise} \end{cases}$$

ConflictFree$_{\mathcal{I}}$ (C) = false

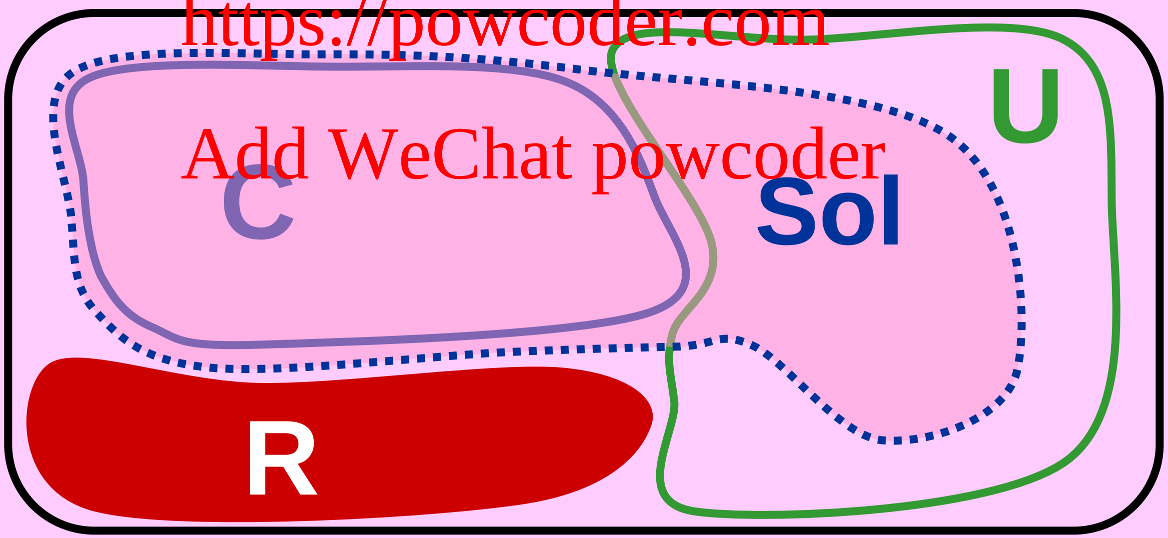
means C has some internal conflict and no matter what additional input objects you add to it, you cannot get a feasible solution that way.
E.g., in any edge-subset of a simple path from A to B, every vertex has in-degree at most 1. So a subset of edges, in which some vertex has in-degree more than 1, has conflict.

The Greedy Loop Invariant

The following are equivalent statements of the generic greedy Loop Invariant:

- The decisions we have made so far are safe, i.e., they are consistent with some optimum solution (if there is any feasible solution).
- Either there is no feasible solution, or there is <u>at least one</u> optimum solution $Sol \in OPT(I)$ that includes every object we have committed to (set C), and excludes every object that we have rejected (set R = S – U – C).

S = Set of all input objects



LI: $FEAS(I) = \emptyset$ or $[\ \exists Sol \in OPT(I) :\ C \subseteq Sol \subseteq C \cup U\ ]$.

Pre-Cond: S is the set of objects in input instance I

Pre-Cond & PreLoopCode ⇒ LI

U ← S
C ← ∅

LI & ¬⟨exit-cond⟩ & LoopCode ⇒ LI
???

Greedy method

LI: FEAS(I) = ∅ or
∃Sol ∈ OPT(I):
C ⊆ Sol ⊆ C ∪ U

Greedy Choice:

select x ∈ U with best

Cost(C∪{x}) possible

U ← U − {x} § permanently decide on x

if ConflictFree(C∪{x}) &

Cost(C∪{x}) better than Cost(C)

then C ← C ∪ {x} § ---------- commit to x

§ else ---------------------------------- reject x

LI & ⟨exit-cond⟩ ⇒ PostLoopCond

U = ∅

NO

YES

FEAS(I) = ∅ or C ∈ OPT(I)

PostLoopCond & PostLoopCode ⇒ Post-Cond

return C

MP = |U|

Post-Cond: output C∈OPT(I) or FEAS(I) = ∅

LI & ¬⟨exit-cond⟩ & LoopCode ⇒ LI

U ≠ ∅ & LI: FEAS(I) = ∅ or ∃Sol ∈ OPT(I) : C ⊆ Sol ⊆ C ∪ U

Case 1:
x committed

Case 2:
x rejected

Greedy Choice:
select x ∈ U with maximum
Cost(C∪{x}) possible

U ← U – {x} § permanently decide on x

if ConflictFree(C∪{x}) &
Cost(C∪{x}) > Cost(C)

then C ← C ∪ {x} § ----------- commit to x

§ else ----------------------------------- reject x

Sol_{new} may or may not be the same as Sol

LI: FEAS(I) = ∅ or ∃Sol_{new} ∈ OPT(I) : C ⊆ Sol_{new} ⊆ C ∪ U

NOTE

LI & ¬⟨exit-cond⟩ & LoopCode ⇒ LI

$U \neq \emptyset$ & LI: $FEAS(I) = \emptyset$ or $\exists Sol \in OPT(I) : C \subseteq Sol \subseteq C \cup U$

Case 1a:
x committed and
x ∈ Sol

OK. Take $Sol_{new} = Sol$

Greedy Choice:
select x ∈ U with maximum
Cost(C∪{x}) possible
U ← U – {x} § permanently decide on x
if ConflictFree(C∪{x}) &
Cost(C∪{x}) > Cost(C)
then C ← C ∪ {x} § ---------- commit to x
§ else --------------------------------- reject x

Sol_{new} may or may not be the same as Sol

LI: $FEAS(I) = \emptyset$ or $\exists Sol_{new} \in OPT(I) : C \subseteq Sol_{new} \subseteq C \cup U$

LI & ¬⟨exit-cond⟩ & LoopCode ⇒ LI

U ≠∅ & LI: FEAS(I) = ∅ or ∃Sol ∈ OPT(I) : C ⊆ Sol ⊆ C ∪ U

Case 1b:
x committed
and
x ∉ Sol

Needs Investigation

Greedy Choice:
select x ∈ U with maximum
Cost(C∪{x}) possible

U ← U – {x} § permanently decide on x
if ConflictFree(C∪{x}) &
Cost(C∪{x}) > Cost(C)

then C ← C ∪ {x} § ---------- commit to x
§ else -------------------------------- reject x

Sol_{new} may or may not be the same as Sol

LI: FEAS(I) = ∅ or ∃Sol_{new} ∈ OPT(I) : C ⊆ Sol_{new} ⊆ C ∪ U

LI & ¬⟨exit-cond⟩ & LoopCode ⇒ LI

$U \neq \emptyset$ & LI: $FEAS(I) = \emptyset$ or $\exists Sol \in OPT(I) : C \subseteq Sol \subseteq C \cup U$

Case 2a:
x rejected and
$x \notin$ Sol

OK. Take
$Sol_{new} = Sol$

Greedy Choice:
select $x \in U$ with maximum
$Cost(C \cup \{x\})$ possible

$U \leftarrow U - \{x\}$ § permanently decide on x

if $ConflictFree(C \cup \{x\})$ &
$Cost(C \cup \{x\}) > Cost(C)$

then $C \leftarrow C \cup \{x\}$ § ---------- commit to x

§ else ---------------------------------- reject x

Sol_{new} may or may not be the same as Sol

LI: $FEAS(I) = \emptyset$ or $\exists Sol_{new} \in OPT(I) : C \subseteq Sol_{new} \subseteq C \cup U$

LI & $\neg\langle$exit-cond$\rangle$ & LoopCode $\Rightarrow$ LI

U $\neq \emptyset$ & LI: FEAS(I) = $\emptyset$ or $\exists$Sol $\in$ OPT(I) : C $\subseteq$ Sol $\subseteq$ C $\cup$ U

Case 2b:
x rejected
and
x $\in$ Sol

Needs Investigation

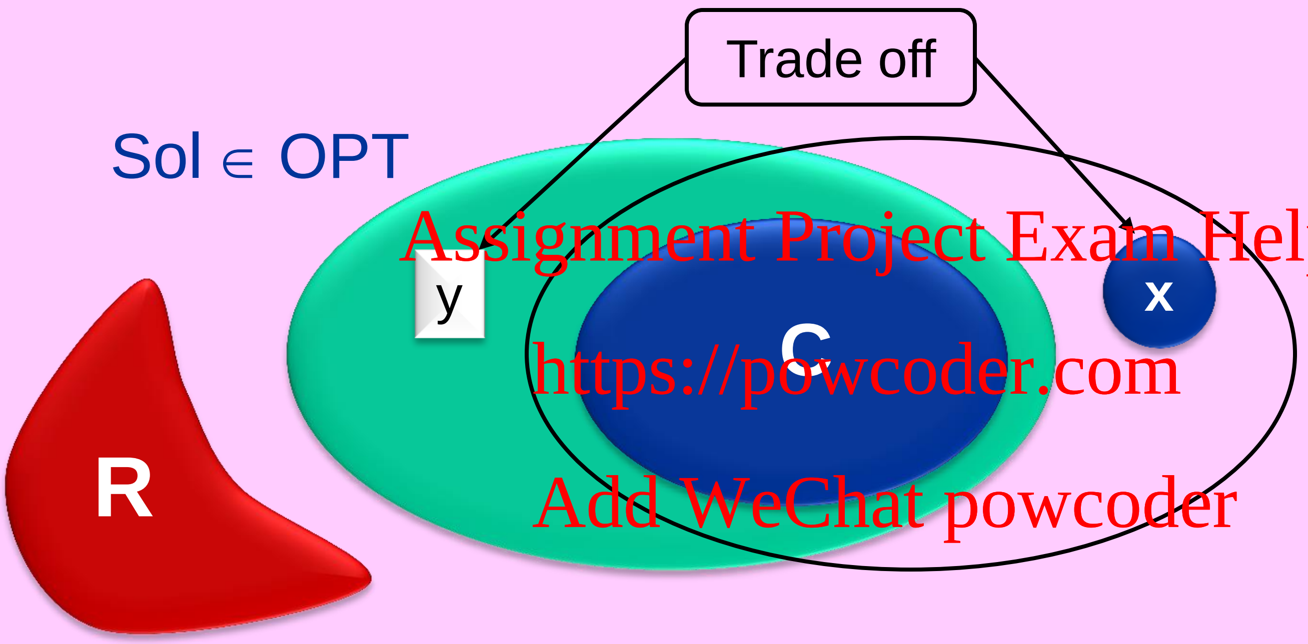
Greedy Choice:
select x $\in$ U with maximum Cost(C$\cup$\{x\}) possible
U $\leftarrow$ U – \{x\}   § permanently decide on x
if ConflictFree(C$\cup$\{x\}) & Cost(C$\cup$\{x\}) > Cost(C)
then C $\leftarrow$ C $\cup$ \{x\}   § ---------- commit to x
§ else ---------------------------------- reject x

Sol_{new} may or may not be the same as Sol

LI: FEAS(I) = $\emptyset$ or $\exists Sol_{new} \in$ OPT(I) : C $\subseteq Sol_{new} \subseteq$ C $\cup$ U

LI & ¬⟨exit-cond⟩ & LoopCode ⇒ LI

Case 1b: x committed and $x \notin$ Sol



Show:
$\exists Sol_{new} \in OPT$

Some times more than one object on either side is traded off.

Show $\exists y \in Sol - C$ such that
$Sol_{new} \leftarrow Sol \cup \{x\} - \{y\}$ satisfies:

(1) $Sol_{new} \in FEAS$, &

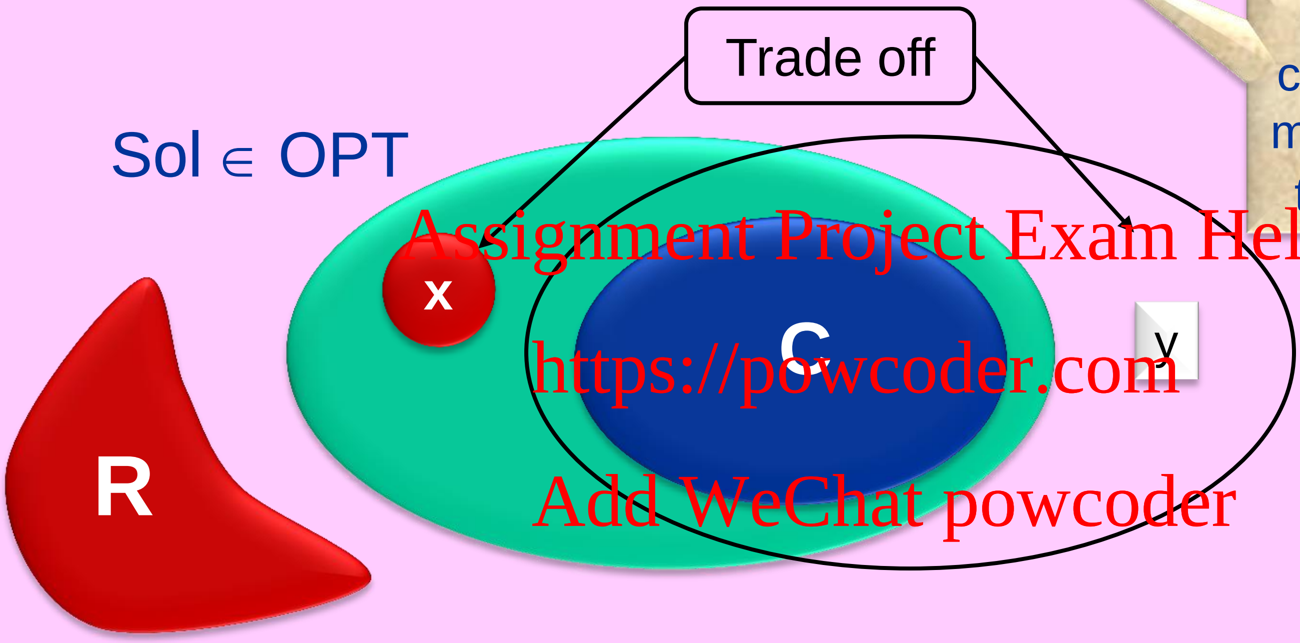
(2) Cost(Sol_{new}) is no worse than Cost(Sol)

LI & $\neg\langle$exit-cond$\rangle$ & LoopCode $\Rightarrow$ **LI**

Case 2b: x rejected and x $\in$ Sol

x could not have created a conflict. So, it must have not improved the objective function.

Sol $\in$ OPT



Show:
$\exists \text{Sol}_{new} \in$ OPT

Some times more than one object on either side is traded off.

Show that
$\text{Sol}_{new} \leftarrow \text{Sol} \cup \{y\} - \{x\}$, for some $y \in U$ - Sol
satisfies:

(1) $\text{Sol}_{new} \in$ FEAS, &

(2) Cost(Sol_{new}) is no worse than Cost(Sol)

COIN CHANGE MAKING

Use minimum number of coins to make change for a given amount.

Greedy: pick the largest coin that fits first, then iterate.

The Coin Change Making Problem

PROBLEM: Within a coin denomination system, make change for a given amount S, using the fewest number of coins possible.

Greedy Strategy (the obvious one):

Commit to the largest coin denomination that fits within the (remaining) amount, then iterate.

Greedy(S) = the greedy solution for amount S.

Optimum(S) = the optimum solution for amount S.

Example 1: Canadian coin denomination system: 25, 10, 5, 1 cents.

S = 98 (Two of many feasible solutions are shown below.)

= 10 + 10 + 10 + 10 + 10 + 10 + 10 + 10 + 10 + 5 + 1 + 1 + 1 ⟵ 13 coins used

= 25 + 25 + 25 + 10 + 10 + 1 + 1 + 1 ⟵ Optimum sol uses 8 coins.

Greedy(98) = Optimum(98) in the Canadian system.

Example 2: Imaginary coin denomination system: 7, 5, 1 kraaps.

S = 10 = 5 + 5 (Optimum) = 7 + 1 + 1 + 1 (Greedy).

Greedy(10) ≠ Optimum(10) in this system.

The Problem Formulation

INPUT: Coin denomination system $a = \langle a_1, a_2, \ldots, a_n \rangle$, $a_1 > a_2 > \ldots > a_n = 1$, and S (all positive integers).

OUTPUT: The solution $x = \langle x_1, x_2, \ldots, x_n \rangle$, where x_t = the number of coin type a_t used, for t = 1..n.

FEAS: $a_1x_1 + a_2x_2 + \ldots + a_nx_n = S$, & x_t is a non-negative integer, for t=1..n.

GOAL: Minimize objective cost $x_1 + x_2 + \ldots + x_n$.

We need $a_n = 1$ to have a feasible sol^n for every S.

$$\underbrace{\text{minimize} \quad x_1 + x_2 + \cdots + x_n}_{\text{objective function}}$$

subject to:

(1) $a_1x_1 + a_2x_2 + \cdots + a_nx_n = S$

(2) $x_t \in \mathcal{N}$, for $t = 1..n$

feasibility constraints

The Greedy Choice & Objective

Greedy Choice: choose the largest coin that fits

Conflict Free: $U \geq 0$ $\quad (U = S - \sum_{i=1}^{n} a_i x_i)$

Problem Objective Cost: $\sum_{i=1}^{n} x_i$

Greedy Objective Cost: $\sum_{i=1}^{n} x_i + \lambda U$ *(λ is an unspecified prohibitively large positive number)*

At the end: $U = 0 \Rightarrow$ Greedy Objective = Problem objective

The Greedy Loop Invariant

Generic Greedy Loop Invariant (∃ feasible solution) :

$$\exists Sol \in OPT : \; C \subseteq Sol \subseteq C \cup U.$$

Current greedy solution: $\langle x_1, x_2, \ldots, x_n \rangle$

Committed to: $\langle x_1, x_2, \ldots, x_t \rangle$, for some $t \in 1..n$.

Rejected: no more of $\langle a_1, a_2, \ldots, a_{t-1} \rangle$.

Considering: any more of a_t?

Not yet considered: $x_k = 0$ for $k > t$.

There is U amount remaining to reach the target value S.

Loop Invariant:

$\langle x_1, x_2, \ldots, x_n \rangle \in FEAS(S - U)$, (need U more to reach target S)

$\exists Sol = \langle y_1, y_2, \ldots, y_n \rangle \in OPT(S)$:

$y_k = x_k$, for $k < t$, (Sol excludes what Greedy has rejected)

$y_t \geq x_t$, (Sol includes what Greedy has committed to)

$x_k = 0$, for $k > t$. (not yet considered)

Algorithm: Greedy Coin Change

Pre-Cond: input is $a = \langle a_1, a_2, \ldots, a_n \rangle$, $a_1 > a_2 > \ldots > a_n = 1$; and S (all pos integers)

Pre-Cond & PreLoopCode $\Rightarrow$ LI

$U \leftarrow S$; $t \leftarrow 1$; $\langle x_1, x_2, \ldots, x_n \rangle \leftarrow \langle 0, 0, \ldots, 0 \rangle$

LI: $\langle x_1, x_2, \ldots, x_n \rangle \in \text{FEAS}(S - U)$,
$\exists \text{Sol} = \langle y_1, y_2, \ldots, y_n \rangle \in \text{OPT}(S)$:
$y_k = x_k$ for $k < t$, $y_t \geq x_t$, $x_k = 0$ for $k > t$.

LI & $\neg\langle$exit-cond$\rangle$ & LoopCode $\Rightarrow$ LI
???

LI & $\langle$exit-cond$\rangle$ $\Rightarrow$ PostLoopCond

$U = 0$ NO / YES

if $U < a_t$
then $t \leftarrow t+1$ § reject: no more a_t
else $x_t \leftarrow x_t + 1$ § commit to another a_t
$U \leftarrow U - a_t$

$\langle x_1, x_2, \ldots, x_n \rangle \in \text{OPT}(S)$

PostLoopCond & PostLoopCode $\Rightarrow$ Post-Cond

return $\langle x_1, x_2, \ldots, x_n \rangle$

Post-Cond: output $\langle x_1, x_2, \ldots, x_n \rangle \in \text{OPT}(S)$

MP = U + (n-t)

Efficient implementation

$$\textit{minimize} \quad x_1 + x_2 + \cdots + x_n$$

objective function

subject to:

$$(1) \quad a_1x_1 + a_2x_2 + \cdots + a_nx_n = S$$

$$(2) \quad x_t \in \mathcal{N}, \text{ for } t = 1..n$$

feasibility constraints

Algorithm Greedy($\langle a_1, a_2, \ldots, a_n \rangle$, S) § takes O(n) time

CoinCount $\leftarrow$ 0; U $\leftarrow$ S

for t $\leftarrow$ 1 .. n **do** § largest coin denomination first

$x_t \leftarrow$ U div a_t § $x_t \leftarrow \lfloor U/a_t \rfloor$

U $\leftarrow$ U mod a_t § $U \leftarrow U - a_t x_t$

CoinCount $\leftarrow$ CoinCount + x_t § objective value

end-for

return ($\langle x_1, x_2, \ldots, x_n \rangle$; CoinCount)

end

Is G(S) = Opt(S) ?

A coin denomination system is called **Regular** if in that system G(S) = Opt(S) **∀S.**

Questions:

(1) In the Canadian Coin System, is G(S) = Opt(S) for all S?

(2) In a given Coin System, is G(S) = Opt(S) for all S?

(3) In a given Coin System, is G(S) = Opt(S) for a given S?

Answers:

(1) YES. It's Regular. We will see why shortly.

(2) YES/NO: Yes if the system is Regular. NO otherwise.
There is a polynomial time Regularity Testing algorithm (described next) to find the YES/NO answer.

(3) YES/NO: Regular system: always yes.
Non-regular system: YES for some S, NO for other S.
Exponential time seems to be needed to find the Yes/No answer.
(This is one of the so called NP-hard problems we will study later.)

How to Test Regularity

- **Question:** Is a given Coin Denomination System Regular,
  i.e., in that system is G(S) = Opt(S) **for all S**?

- Greedy($\langle a_1, a_2, \ldots, a_n \rangle$, S) = ($x = \langle x_1, x_2, \ldots, x_n \rangle$; $G(S) = \Sigma_t x_t$)
  Optimum($\langle a_1, a_2, \ldots, a_n \rangle$, S) = ($y = \langle y_1, y_2, \ldots, y_n \rangle$; $Opt(S) = \Sigma_t y_t$)

- **YES or NO:** For the coin denomination system $\langle a_1, a_2, \ldots, a_n \rangle$:

$\underline{\forall S}$: G(S) = Opt(S).

- There are infinitely many values of S to try.

- However, if there is any counter-example S, there must be one among polynomially many critical values that need to be tested to determine the Yes/No answer. And, each test can be done fast.

  What are these critical values? How are they tested?

Regularity & Critical Values

- Consider the smallest multiple of each coin value that is not smaller than the next larger coin value:

$$S_t = a_t m_t, \qquad m_t = \lceil a_{t-1} / a_t \rceil, \qquad \text{for } t = 2..n.$$

- Necessary Condition for correctness of Greedy:

$$G(S_t) \le m_t, \quad \text{for } t = 2..n. \qquad \text{WHY?}$$

- This also turns out to be sufficient (under some mild pre-condition that is satisfied by virtually any reasonable coin denomination system):

FACT 1: [Regularity Theorem of Magazine, Nemhauser, Trotter, 1975]
Pre-Condition: $S_t < a_{t-2}$, for $t = 3..n$ $\Rightarrow$
$\forall S$: $G(S) = Opt(S)$ $\Leftrightarrow$ $G(S_t) \le m_t$, for $t = 2..n$.

Proof: See Lecture Note 7.

Let's Test the Canadian System

FACT 1: [Magazine, Nemhauser, Trotter, 1975]

Pre-Condition: $S_t < a_{t-2}$, for t = 3..n $\Rightarrow$

$\forall S$: G(S) = Opt(S) $\Leftrightarrow$ $G(S_t) \le m_t$, for t = 2..n.

t	1	2	3	4	5	6
coin a_t	200	100	25	10	5	1
$m_t = \lceil a_{t-1} / a_t \rceil$		2	4	3	2	5
$S_t = a_t\, m_t$		200	100	30	10	5
Greedy: $G(S_t)$		1	1	2	1	1
Pre-Cond: $S_t < a_{t-2}$			yes	yes	yes	yes
Test: $G(S_t) \le m_t$		yes	yes	yes	yes	yes

This table can be constructed and tested in $O(n^2)$ time.

Another System

FACT 1: [Magazine, Nemhauser, Trotter, 1975]

Pre-Condition: $S_t < a_{t-2}$, for $t = 3..n$ $\Rightarrow$

$\forall S$: $G(S) = Opt(S)$ $\Leftrightarrow$ $G(S_t) \le m_t$, for $t = 2..n$.

t	1	2	3	4	5	6
coin a_t	200	100	25	11	5	1
$m_t = \lceil a_{t-1} / a_t \rceil$		2	4	3	3	5
$S_t = a_t m_t$		200	100	33	15	5
Greedy: $G(S_t)$		1	1	5	5	1
Pre-Cond: $S_t < a_{t-2}$			yes	yes	yes	yes
Test: $G(S_t) \le m_t$		yes	yes	NO	NO	yes

This table can be constructed and tested in $O(n^2)$ time.

What if Pre-Condition doesn't hold?

FACT 1: [Magazine, Nemhauser, Trotter, 1975]

Pre-Condition: $S_t < a_{t-2}$, for t = 3..n $\Rightarrow$

$\forall S$: G(S) = Opt(S) $\Leftrightarrow$ $G(S_t) \le m_t$, for t = 2..n.

[This can be tested in $O(n^2)$ time.]

FACT 2: [Pearson, 2005]

$\forall S$: G(S) = Opt(S) $\Leftrightarrow$ $O(n^2)$ critical values test OK in $O(n^3)$ time.

See also Lecture Note 7.

The Optimum Sub-Structure Property

- We just noticed an important property that will be used many times later:
- The optimum sub-structure property: any sub-structure of an optimum structure is itself an optimum structure (for the corresponding sub-instance).
- Problems with this property are usually amenable to more efficient algorithmic solutions than brute-force or exhaustive search methods.
- This property is usually shown by a “cut-&-paste” argument (see below).

- Example 1: The Coin Change Making Problem.
  Consider an optimum solution Sol $\in$ OPT(S). Let G1 be a group of coins in Sol.
  Suppose G1 $\in$ FEAS(U). Then we must have G1 $\in$ OPT(U). Why?
  Because if G1 $\notin$ OPT(U), then we could cut G1 from Sol, and paste in the optimum sub-structure G2 $\in$ OPT(U) instead. By doing so, we would get a new solution Sol’ $\in$ FEAS(S) that has an even better objective value than Sol. But that would contradict Sol $\in$ OPT(S).

- Example 2: The Shortest Path Problem.
  Let P be a shortest path from vertex A to vertex B in the given graph G.
  Let P’ be any (contiguous) sub-path of P. Suppose P’ goes from vertex C to D.
  Then P’ must be a shortest path from C to D. If it were not, then there must be an even shorter path P’’ that goes from C to D. But then, we could replace P’ portion of P by P’’ and get an even shorter path than P that goes from A to B.
  That would contradict the optimality of P.

EVENT SCHEDULING

A banquet hall manager has received a list of reservation requests for the exclusive use of her hall for specified time intervals

She wishes to grant the maximum number of reservation requests that have no time overlap conflicts

Help her select the maximum number of conflict free time intervals

The Problem Statement

INPUT: A set S = { $I_1, I_2, \cdots, I_n$} of n event time-intervals $I_k = \langle s_k , f_k \rangle$, k =1..n,
where s_k = start time of I_k ,
f_k = finishing time of I_k ,
($s_k < f_k$).

OUTPUT: A maximum cardinality subset C $\subseteq$ S of mutually compatible intervals
(i.e., no overlapping pairs).

Example:

S = the intervals shown below,
C = the blue intervals,
|C| =4.

C is not the unique optimum.
Can you spot another optimum solution?

time

Some Greedy Heuristics

Greedy iteration step:

From among undecided intervals, select the interval I that looks BEST.

Commit to I if it's conflict-free
(i.e., doesn't overlap with the committed ones so far).

Reject I otherwise.

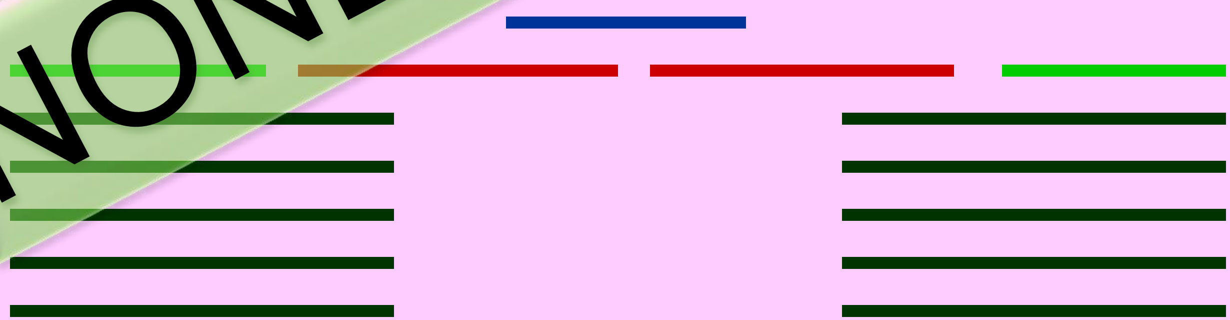
Greedy 1: BEST = earliest start time (min s_k).

Greedy 2: BEST = latest finishing time (max f_k).

Greedy 3: BEST = shortest interval (min $f_k - s_k$).

Greedy 4: BEST = overlaps with fewest # of undecided intervals.

NONE WORKS ☹



Earliest Finishing Time First

S = the set of n given intervals
C = committed intervals
R = rejected intervals
U = undecided intervals

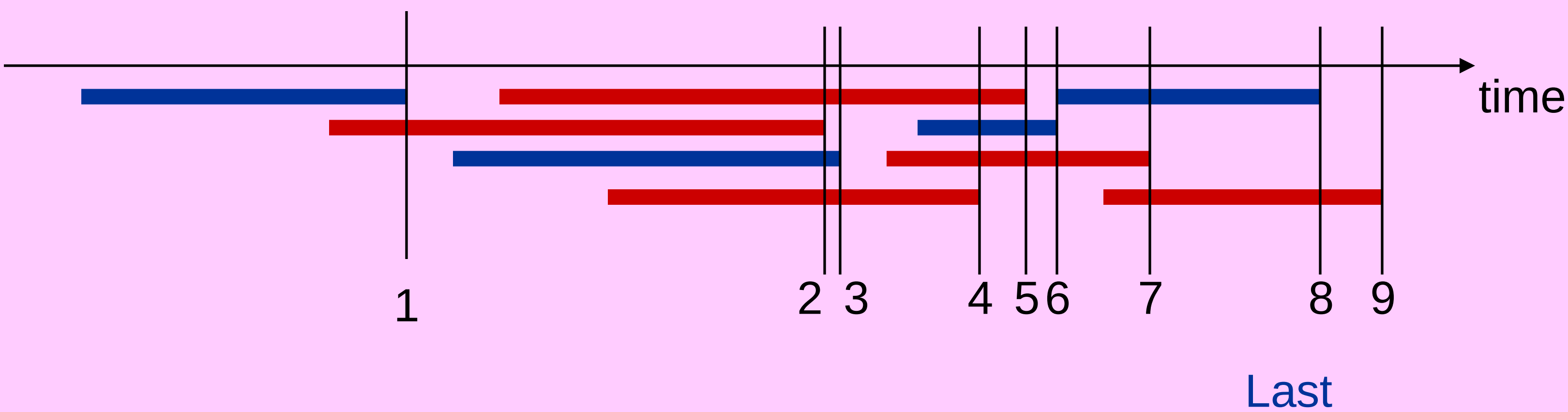
$MaxF(X) = \max\{ f_k \mid I_k \in X \}$
$MinF(X) = \min\{ f_k \mid I_k \in X \}$
$Last = MaxF(C)$

$C \subseteq \{ I_k \in S \mid f_k \le Last \}$
$C \cup R \supseteq \{ I_k \in S \mid f_k < Last \}$
$U \subseteq \{ I_k \in S \mid f_k \ge Last \}$

LOOP INVARIANT:

$$\exists Sol \in Opt(S): \quad C \subseteq Sol \subseteq C \cup U,$$

$$MaxF(C) = Last \le MinF(U).$$



Algorithm: Greedy Event Schedule

Pre-Cond: input is $S = \{I_1, I_2, \cdots, I_n\}$ of n intervals, $I_k = \langle s_k, f_k \rangle$, $s_k < f_k$, $k = 1..n$

$U \leftarrow S$; $C \leftarrow \emptyset$; $Last \leftarrow -\infty$

Pre-Cond & PreLoopCode $\Rightarrow$ LI

LI & $\neg\langle$exit-cond$\rangle$ & LoopCode $\Rightarrow$ LI

???

LI: $\exists Sol \in Opt(S)$:
$C \subseteq Sol \subseteq C \cup U$,
$MaxF(C) = Last \le MinF(U)$

Greedy choice: select $I_k \in U$ with min f_k

$U \leftarrow U - \{I_k\}$ § decide on I_k

if $s_k \ge Last$

then $C \leftarrow C \cup \{I_k\}$ § commit to I_k

$Last \leftarrow f_k$

§ else -------------------------- reject I_k

$U = \emptyset$

NO

YES

LI & $\langle$exit-cond$\rangle$ $\Rightarrow$ PostLoopCond

$C \in OPT(S)$

PostLoopCond & PostLoopCode $\Rightarrow$ Post-Cond

return C

Post-Cond: output $C \in OPT(S)$

MP = |U|

LI & $\neg\langle$exit-cond$\rangle$ & LoopCode $\Rightarrow$ **LI**

$U \neq \varnothing$ &

LI: $\exists Sol \in Opt(S)$: $C \subseteq Sol \subseteq C \cup U$,
$MaxF(C) = Last \leq MinF(U)$

Greedy choice: select $I_k \in U$ with min f_k

$U \leftarrow U - \{I_k\}$ § decide on I_k

if $s_k \geq Last$

then $C \leftarrow C \cup \{I_k\}$ § commit

$Last \leftarrow f_k$

§ else --------------------------- reject

Case 1:
I_k *rejected*

Case 2:
I_k *committed*

Sol_{new} may or may not be the same as Sol

LI: $\exists Sol_{new} \in Opt(S)$: $C \subseteq Sol_{new} \subseteq C \cup U$,
$MaxF(C) = Last \leq MinF(U)$

LI & $\neg\langle$exit-cond$\rangle$ & LoopCode $\Rightarrow$ LI

$U \neq \varnothing$ &

LI: $\exists Sol \in Opt(S)$: $C \subseteq Sol \subseteq C \cup U$,

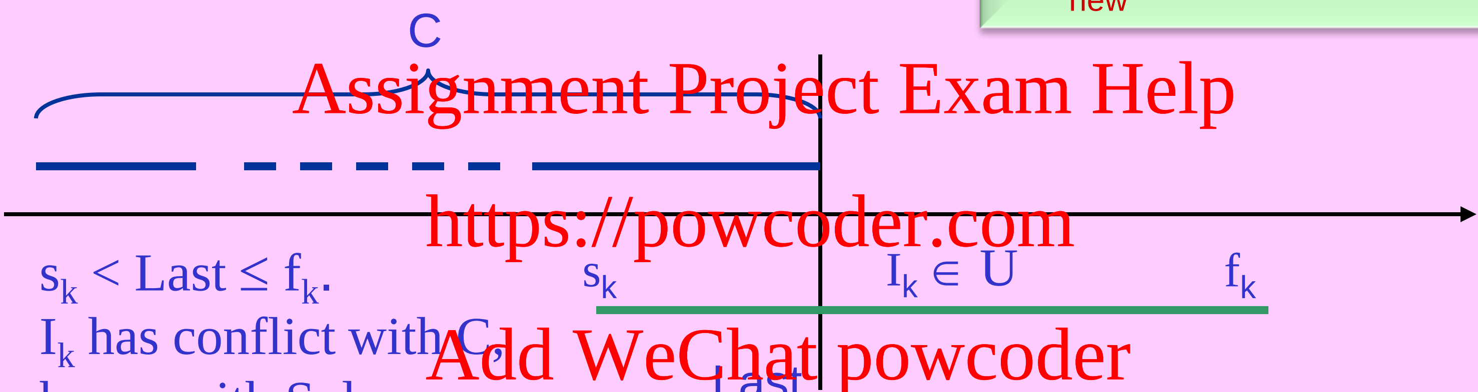
$MaxF(C) = Last \leq MinF(U)$

$U \leftarrow U - \{I_k\}$.

Case 1: $s_k <$ Last

I_k rejected

$Sol_{new} = Sol$ maintains LI.



$s_k < Last \leq f_k$.

I_k has conflict with C,

hence with Sol.

So, $I_k \notin Sol$.

So, $C \subseteq Sol \subseteq C \cup (U - \{I_k\})$. Thus, Sol still satisfies 1st line of the LI.

Removing I_k from U cannot reduce MinF(U).

Also, C and Last don't change.

So, 2nd line of the LI is also maintained.

$\textbf{LI}$ & $\neg\langle$exit-cond$\rangle$ & LoopCode $\Rightarrow$ $\textbf{LI}$

U ≠ ∅ &

LI: $\exists Sol \in Opt(S)$: $C \subseteq Sol \subseteq C \cup U$,

$MaxF(C) = Last \leq MinF(U)$

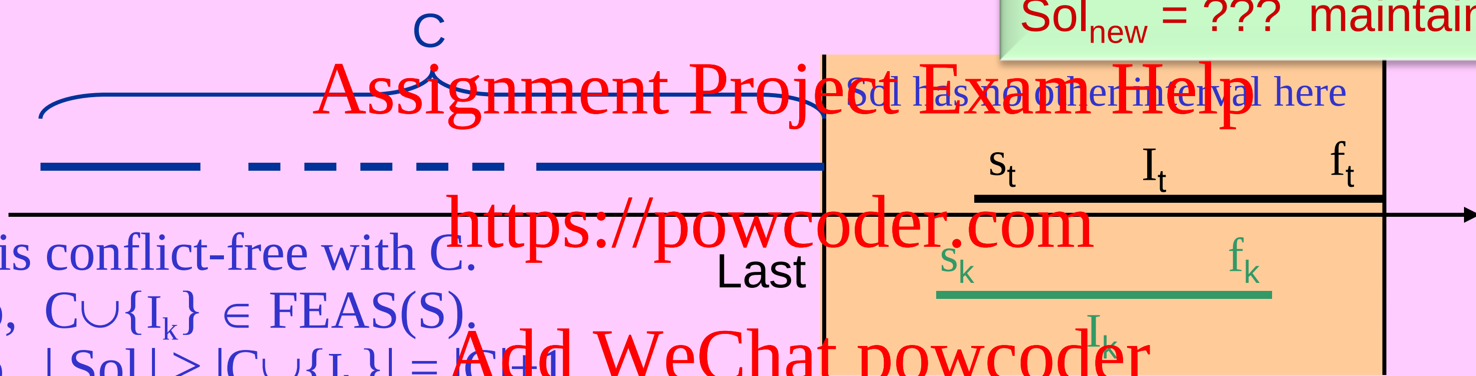
$U \leftarrow U - \{I_k\}$.

Case 2: $s_k \geq$ Last

I_k committed

$(C \leftarrow C \cup \{I_k\};\ Last \leftarrow f_k)$

Sol_{new} = ??? maintains LI.



I_k is conflict-free with C.

So, $C \cup \{I_k\} \in FEAS(S)$.

So, $|Sol| \geq |C \cup \{I_k\}| = |C| + 1$.

So, $\emptyset \neq Sol - C \subseteq U$.

Choose $I_t \in Sol - C$ with min f_t.

$t = k$ or $t \neq k$.

$I_t \in Sol - C \subseteq U$. Greedy choice $\Rightarrow$ $f_t \geq f_k$.

New solution: $\mathbf{Sol_{new} \leftarrow (Sol - \{I_t\}) \cup \{I_k\}}$.

Sol_{new} is conflict-free & $|Sol_{new}| = |Sol|$ $\Rightarrow$ $Sol_{new} \in OPT(S)$.

& $(C \cup \{I_k\}) \subseteq Sol_{new} \subseteq (C \cup \{I_k\}) \cup (U - \{I_k\})$.

Therefore, $\mathbf{Sol_{new}}$ maintains 1st line of the LI.

2nd line of the LI is also maintained.

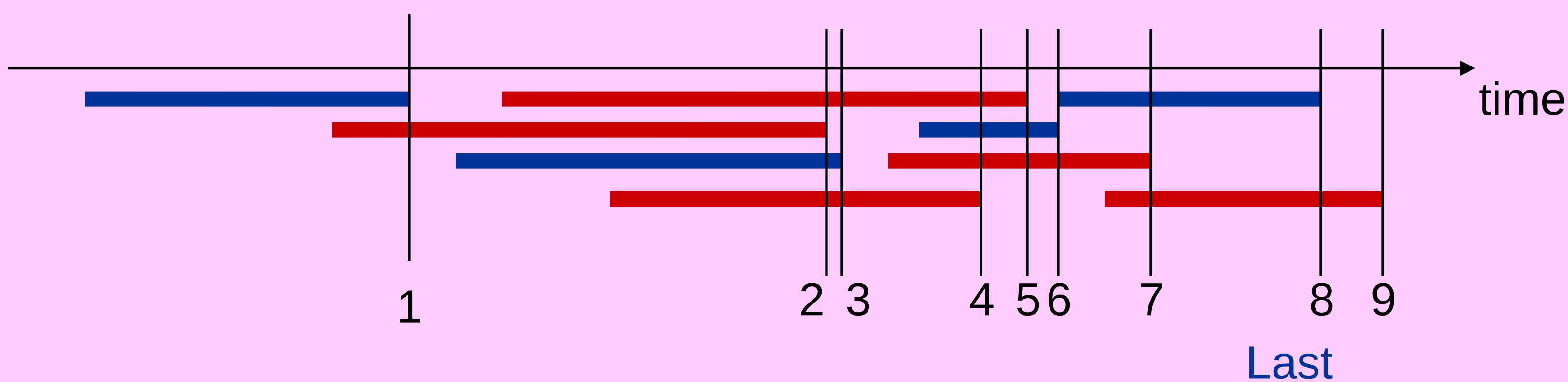
Efficient implementation

Algorithm **GreedyEventSchedule**($S = \langle I_1, I_2, \ldots, I_n \rangle$) § $O(n \log n)$ time

1. SORT S in ascending order of interval finishing times.
   WLOGA $\langle I_1, I_2, \ldots, I_n \rangle$ is the sorted order, i.e., $f_1 \le f_2 \le \ldots \le f_n$.

2. $C \leftarrow \emptyset$; $Last \leftarrow -\infty$
3. **for** $k \leftarrow 1 .. n$ **do**
4. **if** $Last \le s_k$ **then** $C \leftarrow C \cup \{I_k\}$
5. $Last \leftarrow f_k$
6. **end-for**
7. **return** (C)

end



INTERVAL POINT COVER

We have a volunteer group to canvas homes & businesses along Yonge Street.

Each volunteer is willing to canvas a neighboring stretch of houses and shops.

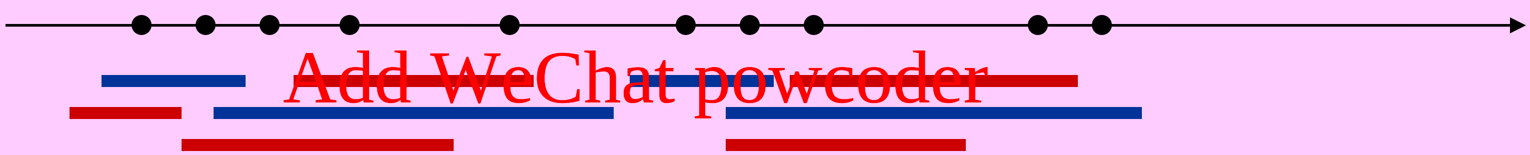
Help us select a minimum number of these volunteers
that can collectively canvas every house and business along the street.

The Problem Statement

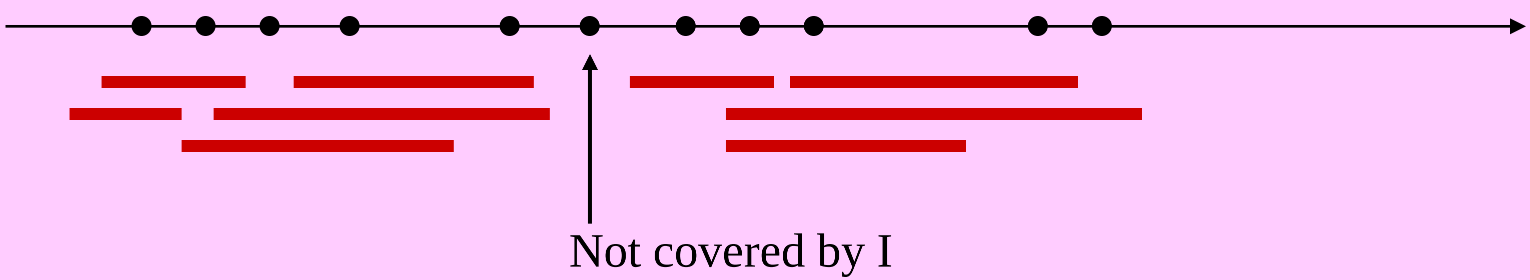
INPUT: A set $P = \{ p_1, p_2, \ldots , p_n\}$ of n points, and
a set $I = \{ I_1= \langle s_1 , f_1\rangle, \; I_2= \langle s_2 , f_2\rangle, \; \ldots , \; I_m= \langle s_m , f_m\rangle \}$ of m intervals,
all on the real line.

OUTPUT: Find out whether or not I collectively covers P.
If yes, then report a minimum cardinality subset $C \subseteq I$ of (possibly overlapping) intervals that collectively cover P.
If not, then report a point $p \in P$ that is not covered by any interval in I.

Example 1:



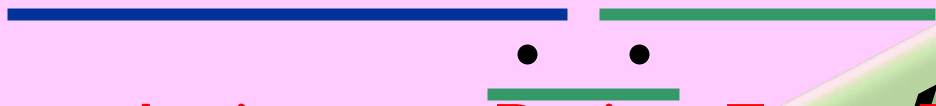
Example 2:



Some Greedy Heuristics

Greedy choice: pick the incrementally BEST undecided interval first.

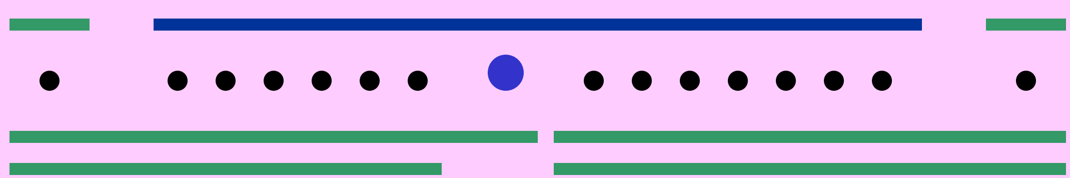
Greedy 1: the longest interval first.



Greedy 2: the interval that covers most uncovered points first.

NONE WORKS ☹

Greedy 3: Let $p \in P$ be an uncovered point that's covered by fewest intervals.
If p is not covered by any interval, then report it.
Otherwise, pick an interval that covers p and max # other uncovered points.



Cover leftmost uncovered point first

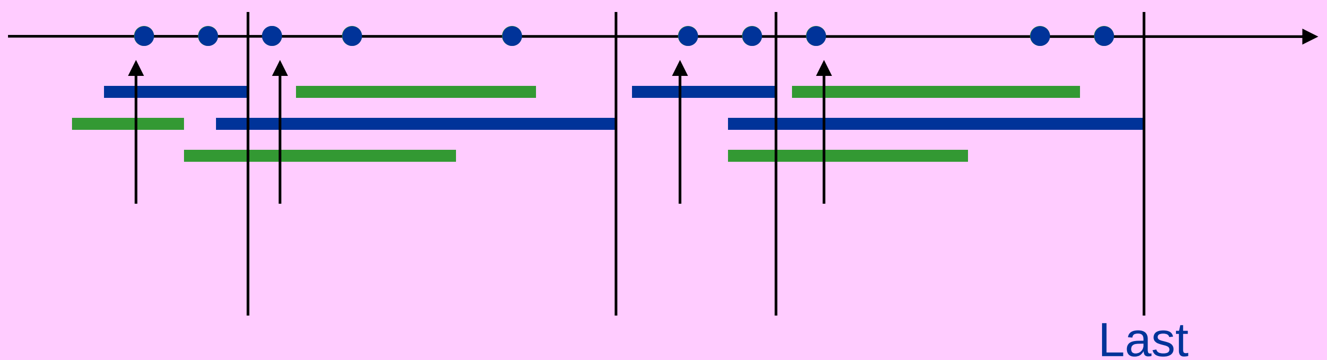
$C \subseteq I$ (committed intervals), $U \subseteq P$ (uncovered points)

<u>GREEDY CHOICE</u>

(1) Pick <u>leftmost</u> point $p \in U$ (not covered by any interval in C).
If no interval in I covers p, then report that p is exposed.
Else, add to C an interval from I – C that covers p and
<u>(2) extends as far right as possible.</u>

<u>LOOP INVARIANT</u>

(exposed $\Rightarrow$ p is not covered by I) &
(I does not cover P or $\exists Sol \in Opt(I,P)$: $C \subseteq Sol$) &
MaxF(C) = Last < Min(U).



Algorithm: Opt Interval Point Cover

Loop Invariant: **(exposed $\Rightarrow$ p is not covered by I) &**
(I does not cover P or $\exists$Sol $\in$ Opt(I,P): C $\subseteq$ Sol) &
MaxF(C) = Last < Min(U).

Algorithm **OptIntervalPointCover** ($P = \{ p_1, p_2, \ldots, p_n\}$, $I = \{I_1, I_2, \ldots, I_m\}$)

$U \leftarrow P$; $C \leftarrow \varnothing$; $Last \leftarrow -\infty$; $exposed \leftarrow$ false
while $U \neq \varnothing$ & not exposed **do**
    $p \leftarrow$ leftmost point in U   § greedy choice 1
    $I' \leftarrow$ set of intervals that cover p
    if $I' = \varnothing$ **then** exposed $\leftarrow$ true
        else do
            Select $I_k \in I'$ with max f_k § greedy choice 2
            $C \leftarrow C \cup \{I_k\}$; $Last \leftarrow f_k$
            $U \leftarrow U - \{q \in P \mid q \leq Last\}$
        end-else
end-while
if exposed **then return** (p is not covered by I)
        else return (C)
end

PreLoopCode: $U \leftarrow P$; $C \leftarrow \varnothing$; $Last \leftarrow -\infty$; $exposed \leftarrow false$

⟨exit-cond⟩: $U = \varnothing$ or exposed.

LI: (exposed $\Rightarrow$ p is not covered by I) &
(I does not cover P or $\exists Sol \in Opt(I,P): C \subseteq Sol$) &
$MaxF(C) = Last < Min(U)$.

Pre-Cond & PreLoopCode $\Rightarrow$ LI

Lines 1, 2, 3 of the LI become true.

LI & ⟨exit-cond⟩ $\Rightarrow$ PostLoopCond

Case 1. [exposed = true]:

exposed $\Rightarrow$ p is not covered by I. (1st line of LI)

$\therefore$ p is not covered by I.

Case 2. [$U = \varnothing$ & exposed = false]:

C covers P. (3rd line of LI)

$C \subseteq Sol \Rightarrow C \in Opt(I,P)$. (2nd line of LI)

$\therefore$ C is an optimum cover.

LI & ¬⟨exit-cond⟩ & LoopCode ⇒ LI

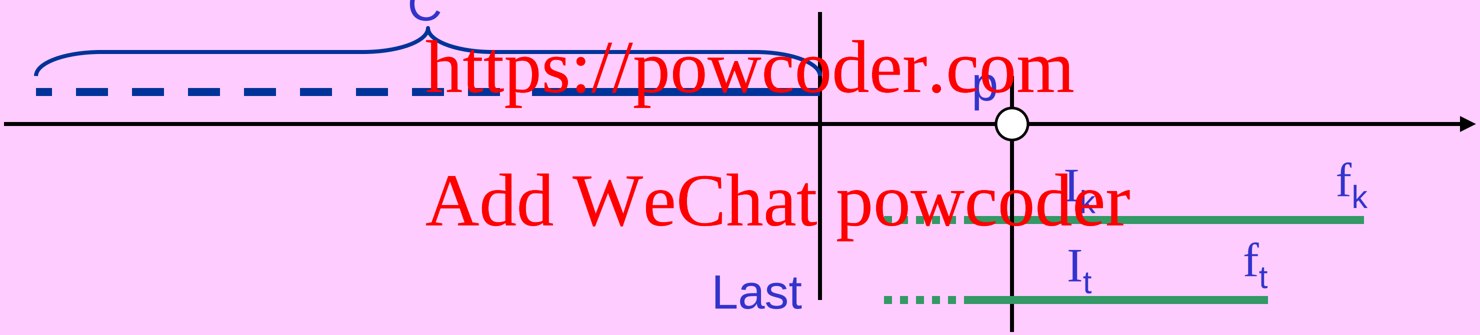
$U \neq \emptyset$ & not exposed &

LI: (exposed ⇒ p is not covered by I) &
(I does not cover P or
$\exists$Sol $\in$ Opt(I,P): C $\subseteq$ Sol) &
MaxF(C) = Last $\leq$ Min(U).

Case 2: p covered by $I_k \in I'$, max f_k

$C \leftarrow C \cup \{I_k\}$; Last $\leftarrow f_k$
$U \leftarrow U - \{q \in P \mid q \leq \text{Last}\}$

Sol_{new} = ??? maintains LI.



Suppose $I_t \in$ Sol – C covers p. So, $I_t \in I'$. We have $t = k$ or $t \neq k$.
$f_t \leq f_k$ (by greedy choice 2).
New solution: $Sol_{new} \leftarrow (Sol - \{I_t\}) \cup \{I_k\}$.
Sol_{new} covers every point of P covered by Sol , and $|Sol_{new}| = |Sol|$.
Therefore, Sol $\in$ OPT(I,P) ⇒ $Sol_{new} \in$ OPT(I,P).
$C \cup \{I_k\} \subseteq Sol_{new}$. Therefore, Sol_{new} still maintains 3rd line of the LI.
Remaining lines of LI are also maintained.

Efficient Implementation

To carry out the greedy choices fast:

- Line-Scan critical event times t left-to-right on the real line.

- Classify each interval $I_k = \langle s_k , f_k \rangle \in I$:

  Inactive: $t < s_k$ (t hasn't reached I_k)

  Active: $s_k \le t \le f_k$ (t is covered by I_k) $\Big\}$ Activated.

  Dead: $f_k < t$ (t has passed I_k)

- Classify event e:

  $e \in P$ point event (point activation time = p)

  $e=(s_k , I_k)$ interval activation event (interval activation time = s_k)

- **ActInts** = <u>Max</u> Priority Queue of activated (= active/dead) intervals I_k [priority = f_k].

- **Events** = <u>Min</u> Priority Queue of unprocessed events [priority = activation time].

- **Iterate:** $e \leftarrow$ DeleteMin(Events); process event e.

- **Minor trick:** to avoid DeleteMax on empty ActInts, insert as first activated event a dummy interval $I_0 = \langle -\infty , -\infty \rangle$. I_0 will remain in ActInts till the end.

Algorithm: Efficient implementation

Algorithm **OptIntervalPointCover** ($P = \{ p_1, p_2, \ldots, p_n\}$, $I= \{I_1, I_2, \ldots, I_m\}$)

```
C ← ∅;   Last ← – ∞;   I₀ ← ⟨ – ∞ , – ∞ ⟩;   I ← I ∪ {I₀}
MakeEmptyMaxHeap(ActInts)
Events ← ConstructMinHeap(P ∪ I)                          § O(n+m) time
while Events ≠ ∅ do                                        § O(n + m) iterations
    e ← DeleteMin(Events)          § next event to process, O(log(n+m)) time
    if e = (s_k , I_k) then Insert(I_k , ActInts)          § activate interval
        else                                               § event e is a point in P
        if e > Last then do       § greedy choice 1: e = leftmost uncovered point
            I_k ← DeleteMax(ActInts)     § greedy choice 2, O(log m) time
            if f_k < e then return ( point e∈P is not covered by I )
                      else do
                          C ← C ∪ {I_k}
                          Last ← f_k
                      end-else
        end-if
end-while
return ( C )
end
```

O((n+m) log(n+m)) time

Bibliography

If you want to dig deeper, roots of greedy algorithms are in the theory of matroids:

- Hassler Whitney, *"On the abstract properties of linear dependence,"* American Journal of Mathematics, 57:509-533, 1935.
- Jack Edmonds, *"Matroids and the greedy algorithm,"* Mathematical Programming, 1:126-136, 1971.
- Eugene L. Lawler, *"Combinatorial Optimization: Networks and Matroids,"* Holt, Rinehart, and Winston, 1976.
- Christos Papadimitriou and Kenneth Steiglitz, *"Combinatorial Optimization: Algorithms and Complexity,"* Prentice-Hall, 1982. [2nd edition, Courier Dover (publisher), 1998.]

The two cited references on the coin change making problem are:

- M.J. Magazine, G.L. Nemhauser, L.E. Trotter, Jr., *"When the greedy solution solves a class of knapsack problems,"* Operations Research, 23(2):207-217, 1975.
- David Pearson *"A polynomial-time algorithm for the change-making problem,"* Operations Research Letters, 33(3):231-234, 2005.

Exercises

1. **The shortest obstacle avoiding path:** As we discussed, the scene consists of a pair of points A and B among n pairwise disjoint rectangular obstacles with horizontal and vertical sides. We listed 3 greedy heuristics. We saw that all three fail to find the shortest obstacle avoiding A-B path on some instances.
   An interesting question arises: How badly can these heuristics fail?
   (a) Explore to find the worst scene for each of these heuristics. By worse, we mean the ratio of the length of the path found by the heuristic compared with the length of the shortest path, expressed as a function of n, is as high as possible. How bad could it be? Is it unbounded? Supper-linear? Linear? Sub-linear? …
   (b) Does the answer improve if the obstacles are congruent squares?

2. **Coin Change making:** For each of the following coin denomination systems either argue that the greedy algorithm always yields an optimum solution for any given amount, or give a counter-example:

   (a) Coins c^0 , c^1 , c^2 , …, c^{n-1}, where c is an integer > 1.
   (b) Coins 1, 7, 13, 19, 61.

   (c) Coins 1, 7, 14, 20, 61.

3. **[CLRS, Exercise 16.2-5, page 428] Smallest unit length interval covering set:** Describe an efficient algorithm that, given a set P = { p_1, p_2, … , p_n} of points on the real line, determines the smallest set of unit-length closed intervals that covers all of the given points. Argue that your algorithm is correct.

4. **Interval Point Cover:** What is the time complexity of the Interval Point Cover problem? We developed an O((n+m) log (n+m)) time algorithm for this problem. Derive a lower bound on this problem by a reduction from the Min-Gap Problem.

5. **[CLRS, Exercise 16.1-4, page 379] Minimum number of lecture halls:** Suppose we have a set of events to schedule among a large number of lecture halls. We wish to schedule all the events using as few lecture halls as possible. Design and analyze an efficient greedy algorithm to determine which event should use which lecture hall. Prove the optimality of the solution produced by your algorithm.

[This is also known as the **interval-graph colouring problem**. We can create an interval graph whose vertices are the given events and whose edges connect incompatible events. The smallest number of colours required to colour every vertex so that no two adjacent vertices are given the same colour corresponds to finding the fewest lecture halls needed to schedule all the given events.]

6. **Smallest Hitting Set:** Design and analyze an efficient greedy algorithm for the following problem:

Input: A set $P = \{ p_1, p_2, \ldots, p_n \}$ of points, and a set $I = \{ I_1, I_2, \ldots, I_m \}$ of intervals, all on the real line. These intervals and points are given in no particular order. Each interval is given by its starting and finishing times.

Output: (i) A minimum cardinality subset H of P such that every interval in I is hit by (i.e., contains) at least one point in H, or
(ii) an interval $I_k \in I$ that is not hit by any point in P.

7. Given a set of black and white intervals, select a smallest number of white intervals that collectively overlap every black interval. State your greedy choice and prove its correctness.

8. **One Machine Scheduling with Deadlines:**
You are given a set $\{J_1, J_2, \ldots, J_n\}$ of n jobs to be processed on a single sequential machine. Associated with each job J_k is a processing time t_k and a deadline d_k by which it must be completed. A feasible schedule is a permutation of the jobs such that if the jobs are processed in that order, then each job finishes by its deadline.
Design & analyze a simple greedy strategy that finds a feasible schedule if there is any.

9. **Two Machine Scheduling:**
We are given a set $\{J_1, J_2, \ldots, J_n\}$ of n jobs that need to be processed by two machines A and B. These machines perform different operations and each can process only one job at a time. Each job has to be processed by both machines; first by A, then by B.
Job J_k requires a given duration A_k on machine A, and a given duration B_k on B.

We wish to find the minimum total duration required to process all n jobs by both machines, as well as the corresponding optimum schedule.
A schedule is the sequencing of the n jobs through the two machines.
Both machines will process the jobs based on the scheduled order. Each job J, in the scheduled order, is first processed on machine A as soon as A completes its previously scheduled job. Upon completion by A, job J is processed by B as soon as B becomes available.

Design and analyze an efficient greedy algorithm for this problem.
Prove the optimality of the schedule produced by your algorithm.

10. The widely popular Spanish search engine **El Goog** needs to do a serious amount of computation every time it recompiles its index. Fortunately, the company has at its disposal a single large supercomputer, together with an essentially unlimited supply of high-end PCs.

They have broken the overall computation into n distinct jobs, labeled $J_1, J_2, \ldots, J_n$, which can be performed completely independently of one another. Each job consists of two stages: first it needs to be **preprocessed** on the supercomputer, and then it needs to be **finished** on one of the PCs. Let's say that job J_k needs p_k seconds of time on the supercomputer, followed by f_k seconds of time on a PC.

Since there are at least n PCs available on the premises, the finishing of the jobs can be performed fully in parallel - all the jobs can be processed at the same time. However, the supercomputer can only work on a single job at a time. So the system managers need to work out an order in which to feed the jobs to the supercomputer. As soon as the first job in order is done on the supercomputer, it can be handed off to a PC for finishing; at that point in time a second job can be fed to the supercomputer; when the second job is done on the supercomputer, it can proceed to a PC regardless of whether or not the first job is done (since PCs work independently in parallel), and so on.

Let's say that a **schedule** is an ordering of the jobs for the supercomputer, and the **completion time** of the schedule is the earliest time at which all jobs will have finished processing on the PCs. This is an important quantity to minimize, since it determines how rapidly El Goog can generate a new index.

Design and analyze an efficient greedy algorithm to find a minimum completion time schedule.

11. **The Factory Fueling Problem:** A remotely-located factory has a fuel reservoir which, when full, has enough fuel to keep the factory's machines running for M days. Due to the factory's remote location, the fuel supply company does not deliver fuel on-demand. Instead, its trucks make visits to the factory's area according to a preset annual schedule and if requested, would fill the reservoir. The schedule is given as an array S[1..n] where S[i] is the date of the i-th visit in the year. Each time the reservoir is filled, a fixed delivery charge is applied regardless of how much fuel is supplied. The factory management would like to minimize these delivery charges and, at the same time, minimize the idle, empty reservoir periods. Given M and S, describe an efficient greedy algorithm to obtain an optimal annual filling schedule; i.e., determine for each of the n visits whether the reservoir should be filled. Prove that your algorithm satisfies the greedy-choice property.

12. **The Loading Problem:** Consider a train that travels from station 1 to station n with intermediate stops at stations 2, 3, …, (n - 1). We have p[i,j] packages that need to be delivered from point i to point j where $1 \le i < j \le n$. Packages have the same size. The maximum number at any point in the train must not exceed its capacity C. We want to deliver as many packages as possible.

    a) Give a strategy for dropping and picking up packages at each point.
    b) Prove your strategy maximizes the number of delivered packages.

    [Hint: Use the greedy idea twice in the solution of this problem. At point 1 load packages in increasing order of their destination till capacity is reached. When the train arrives at point 2, (conceptually) unload and reload according to the same principle as above. If some packages that were picked up at point 1 get left at point 2, do not load them at point 1 in the first place!]

13. Compatible Capacitated Assignment:

Input: Two sorted arrays A[1..n] and B[1..n] of reals; a positive integer capacity C, and a positive Compatibility real number α.

Definitions:

(i) A pair (i,j) is called **compatible**, if $|A[i] - B[j]| \le \alpha$.

(ii) An **assignment** is a pairing among elements of arrays A and B so that each element of each array appears in at least one pairing with an element of the other array.

(iii) An assignment is **compatible** if each pairing in that assignment is compatible.

(iv) A **valid** assignment is a compatible one in which no element is paired with more than C elements from the other array.

Output: A valid assignment, or nil if no such assignment exists.

Design analyze and carefully prove the correctness of an efficient greedy algorithm for this problem. [Hint: Show the following claims are true:

(i) If (i,j_1) and (i,j_2) are compatible, then so is every pair (i,j) such that j is between j_1 and j_2. Similarly, if (i_1,j) and (i_2,j) are compatible, then so is every pair (i,j) such that i is between i_1 and i_2.

(ii) If there is a valid assignment, then there is a valid assignment with no crossing, where a crossing consists of two assigned pairs (i_1,j_1) and (i_2,j_2) such that $i_1 < i_2$ but $j_1 > j_2$, or $i_1 > i_2$ but $j_1 < j_2$.

(iii) An uncrossed valid assignment has the property that if (i,j_1) and (i,j_2) are pairs in that assignment, then so are all pairs of the form (i,j) with j between j_1 and j_2. Similarly, if (i_1,j) and (i_2,j) are pairs in the assignment, then so are all pairs (i,j) such that i is between i_1 and i_2.

(iv) We can assign pairs by scanning arrays A and B in a greedy merge fashion.]

14. Egyptian Fractions:

Input: A rational fraction x/y, where x and y are integers and $0<x<y$,

Output: Express x/y as the sum $1/m_1 + 1/m_2 + \cdots + 1/m_k$, where m_i are positive integers and minimize k. That is, the sum of a minimum number of unit fractions.

This is always possible since we can express x/y as the sum of x copies of $1/y$. So, minimum $k \le x$. The answer is not always unique, e.g., $2/3$ can be expressed as $2/3 = 1/2 + 1/6 = 1/3 + 1/3$.

A wrong greedy strategy: first pick the largest unit fraction that fits.

$$4/17 = 1/5 + 1/29 + 1/1233 + 1/3039345 \quad \text{(Greedy)}$$
$$= 1/6 + 1/15 + 1/510 \quad \text{(Optimum)}$$

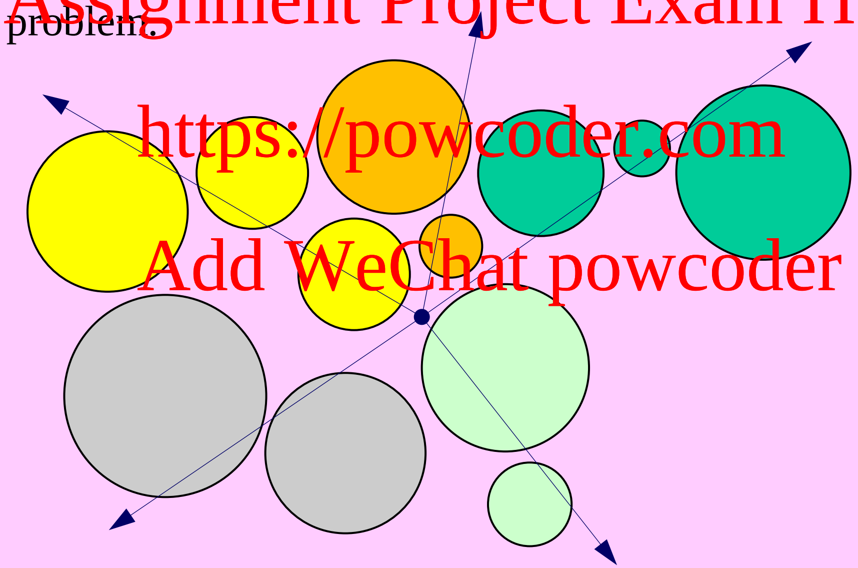
a) Give a correct greedy strategy for this problem.

b) Prove that your strategy indeed minimizes k.

c) Are any of the following conjectures true? Why?
Erdos-Straus Conjecture: $4/N$ can be written as sum of up to 3 unit fractions.
Sierpinski Conjecture: $5/N$ can be written as sum of up to 3 unit fractions.
(N is an arbitrary positive integer.)

15. Laser Beams and Balloons:

Suppose you have a high-powered laser beam gun in your hand, standing amidst a number of large balloons in a vast open plateau. You want to use your laser gun to shoot all the balloons without moving from your current location. You want to fire as few shots as possible. The (2 dimensional) *Minimum Beams Problem (MBP)* can be stated more formally as follows. Given a set C of n circles in the plane, each specified by its radius and the (x, y) coordinates of its center, compute a minimum number of laser beam directions from the origin that collectively intersect every circle in C. [Each such non-vertical beam may geometrically be viewed as a half-line with the equation $y = sx$ (with slope s) that is restricted to one of the 4 quadrants by an additional inequality $x \geq 0$ or $x \leq 0$.] Your goal is to find an efficient algorithm for this problem.



12 balloons hit by 5 laser beams.

[Problem description continued on the next page.]

15. Laser Beams and Balloons (continued):

a) For the sake of this problem we will assume, that in addition to the RAM arithmetic operations, the square-root operation on reals also takes $O(1)$ time. Give a list of geometric primitives that would be needed by your algorithm. These operations, when implemented, should run in $O(1)$ time. The details of their implementation is left as optional (not required). However, you must give a precise definition of these primitives. For instance, the boolean function $Intersects(b, c)$ returns true if and only if beam b intersects circle c. You should convince yourself that this function can be implemented to run in $O(1)$ time.

b) First consider the special case of MBP where there is a known beam direction that does not hit any circle. Can you transform this special case to a more familiar problem and efficiently obtain the optimum solution that way?

c) Using part (b), describe an efficient algorithm that *approximately* solves the general MBP by producing a solution that is either optimum or uses one more beam than the optimum. Prove that fact.

d) Design and analyze an $O(n^2)$ time algorithm that *exactly* solves the general MBP. You must prove that your algorithm does indeed output an *optimum* solution. [Hint: each circle can in turn be considered as the "first" candidate target. How do you proceed from there to handle the rest?]

e) [10% Extra Credit and Optional]
Design and analyze an $O(n \log n)$ time algorithm that *exactly* solves the general MBP. Don't forget the optimality proof. [Hint: This is more challenging!]

END