Algorithms and datastructures II

Lecture 10: NP-completeness

Jan Hubička

Department of Applied Mathematics
Charles University
Prague

Dec 7 2020

Decision problems

Definition

A (decision) problem if a function from $\{0,1\}^*$ (the set of all possible inputs) to $\{0,1\}$.

Definition (Reduction)

Given problems A and B, we say that A is (polynomial time) reducible to B (and write $A \longrightarrow B$) if there exists function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ such that for every $x \in \{0,1\}^*$ it holds $A(x) = B(f(x))$ and f can be computed in polynomial time relative to $|x|$. Function f is also called (polynomial time) reduction.

Decision problems

Definition

A (decision) problem if a function from $\{0,1\}^*$ (the set of all possible inputs) to $\{0,1\}$.

Definition (Reduction)

Given problems A and B, we say that A is (polynomial time) reducible to B (and write $A \longrightarrow B$) if there exists function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ such that for every $x \in \{0,1\}^*$ it holds $A(x) = B(f(x))$ and f can be computed in polynomial time relative to $|x|$. Function f is also called (polynomial time) reduction.

"I can't find an efficient algorithm, but neither can all these famous people."

Definition (P)

P is the class of all (decision) problems that can be solved by a polynomial time algorithm.

$L \in P$ if and only if there exists algorithm A and polynomial f such that for every input x running $A(x)$ will finish in time at most $f(|x|)$ and $A(x) = L(x)$.

Definition (P)

P is the class of all (decision) problems that can be solved by a polynomial time algorithm.

$L \in P$ if and only if there exists algorithm A and polynomial f such that for every input x running $A(x)$ will finish in time at most $f(|x|)$ and $A(x) = L(x)$.

Definition (NP)

NP is the class of all (decision) problems L such that there exists some problem $K \in P$ and a polynomial g such that for every input x it holds that $L(x) = 1$ iff there exists $y \in \{0,1\}^*$ of length at most $g(|x|)$ such that $K(x, y) = 1$.

You can think of y as a certificate that $L(x) = 1$.

Definition (P)

P is the class of all (decision) problems that can be solved by a polynomial time algorithm.

$L \in P$ if and only if there exists algorithm A and polynomial f such that for every input x running $A(x)$ will finish in time at most $f(|x|)$ and $A(x) = L(x)$.

Definition (NP)

NP is the class of all (decision) problems L such that there exists some problem $K \in P$ and a polynomial g such that for every input x it holds that $L(x) = 1$ iff there exists $y \in \{0, 1\}^*$ of length at most $g(|x|)$ such that $K(x, y) = 1$.

You can think of y as a certificate that $L(x) = 1$.

Observation

SAT $\in NP$.

Definition (P)

P is the class of all (decision) problems that can be solved by a polynomial time algorithm.

$L \in P$ if and only if there exists algorithm A and polynomial f such that for every input x running $A(x)$ will finish in time at most $f(|x|)$ and $A(x) = L(x)$.

Definition (NP)

NP is the class of all (decision) problems L such that there exists some problem $K \in P$ and a polynomial g such that for every input x it holds that $L(x) = 1$ iff there exists $y \in \{0,1\}^*$ of length at most $g(|x|)$ such that $K(x, y) = 1$.

You can think of y as a certificate that $L(x) = 1$.

Observation

SAT $\in NP$.

Observation

$P \subseteq NP$.

Definition (P)

P is the class of all (decision) problems that can be solved by a polynomial time algorithm.

$L \in P$ if and only if there exists algorithm A and polynomial f such that for every input x running $A(x)$ will finish in time at most $f(|x|)$ and $A(x) = L(x)$.

Definition (NP)

NP is the class of all (decision) problems L such that there exists some problem $K \in P$ and a polynomial g such that for every input x it holds that $L(x) = 1$ iff there exists $y \in \{0,1\}^*$ of length at most $g(|x|)$ such that $K(x, y) = 1$.

You can think of y as a certificate that $L(x) = 1$.

Observation

SAT $\in NP$.

Observation

$P \subseteq NP$.

$P = NP$ is open since 1970's.

NP-completeness

Definition (NP-hardness)

Problem L is NP-hard if every problem from NP can be reduced to L.

Lemma

If some NP-hard problem L is in P then $P = NP$.

NP-completeness

Definition (NP-hardness)

Problem L is NP-hard if every problem from NP can be reduced to L.

Lemma

If some NP-hard problem L is in P then $P = NP$.

Definition (NP-completeness)

NP-hard problem L is called NP-complete iff it is in P.

Theorem (Cook's theorem or Cook-Levin theorem)

SAT is NP-complete.

NP-completeness

Definition (NP-hardness)

Problem L is NP-hard if every problem from NP can be reduced to L.

Lemma

If some NP-hard problem L is in P then $P = NP$.

Definition (NP-completeness)

NP-hard problem L is called NP-complete iff it is in P.

Theorem (Cook's theorem or Cook-Levin theorem)

SAT is NP-complete.

Lemma

Given two problems $L, M \in NP$. If L is NP-complete and $L \longrightarrow M$, then M is NP-complete.

NP-completeness

Definition (NP-hardness)

Problem L is NP-hard if every problem from NP can be reduced to L.

Lemma

If some NP-hard problem L is in P then $P = NP$.

Definition (NP-completeness)

NP-hard problem L is called NP-complete iff it is in P.

Theorem (Cook's theorem or Cook-Levin theorem)

SAT is NP-complete.

Lemma

Given two problems $L, M \in NP$. If L is NP-complete and $L \longrightarrow M$, then M is NP-complete.

Examples of NP-complete problems:

1. Logical problems: SAT, 3-SAT, 3,3-SAT, SAT for general formulas (not in CNF), Boolean CircuitSAT, . . .

NP-completeness

Definition (*NP*-hardness)

Problem L is NP-hard if every problem from NP can be reduced to L.

Lemma

If some NP-hard problem L is in P then $P = NP$.

Definition (*NP*-completeness)

NP-hard problem L is called NP-complete iff it is in P.

Theorem (Cook's theorem or Cook-Levin theorem)

SAT is NP-complete.

Lemma

Given two problems $L, M \in NP$. If L is NP-complete and $L \longrightarrow M$, then M is NP-complete.

Examples of NP-complete problems:

1. Logical problems: SAT, 3-SAT, 3,3-SAT, SAT for general formulas (not in CNF), Boolean CircuitSAT, . . .
2. Graph problems: IndSet, Clique, graph coloring, Hamiltonian path, Hamiltonian cycle, . . .

NP-completeness

Definition (NP-hardness)

Problem L is NP-hard if every problem from NP can be reduced to L.

Lemma

If some NP-hard problem L is in P then $P = NP$.

Definition (NP-completeness)

NP-hard problem L is called NP-complete iff it is in P.

Theorem (Cook's theorem or Cook-Levin theorem)

SAT is NP-complete.

Lemma

Given two problems $L, M \in NP$. If L is NP-complete and $L \longrightarrow M$, then M is NP-complete.

Examples of NP-complete problems:

1. Logical problems: SAT, 3-SAT, 3,3-SAT, SAT for general formulas (not in CNF), Boolean CircuitSAT, . . .
2. Graph problems: IndSet, Clique, graph coloring, Hamiltonian path, Hamiltonian cycle, . . .
3. Numerical problems: Finding subset of a given sum, Knapsack, $Ax = 1$, . . .

Stephen Cook, Leonid Levin

Cook-Levin theorem, 1971

Theorem (Cook's theorem or Cook-Levin theorem)

SAT is NP-complete.

Direction of attack:

1. Show that every problem in NP can be solved by CircuitSAT
2. Show reduction of CircuitSAT to SAT

Cook-Levin theorem, 1971

Theorem (Cook's theorem or Cook-Levin theorem)

SAT is NP-complete.

Direction of attack:

1. Show that every problem in NP can be solved by CircuitSAT
2. Show reduction of CircuitSAT to SAT

Lemma

Let L be a problem in P. Then there exists polynomial p and an algorithm which in time $p(n)$ builds boolean circuits B_n with n inputs and one output that solves L.
(For every $x \in \{0,1\}^*$ it holds that $B_n(x) = L(x)$.)

Cook-Levin theorem, 1971

Theorem (Cook's theorem or Cook-Levin theorem)

SAT is NP-complete.

Direction of attack:

1. Show that every problem in NP can be solved by CircuitSAT
2. Show reduction of CircuitSAT to SAT

Lemma

Let L be a problem in P. Then there exists polynomial p and an algorithm which in time $p(n)$ builds boolean circuits B_n with n inputs and one output that solves L.
(For every $x \in \{0,1\}^*$ it holds that $B_n(x) = L(x)$.)

Proof (sketch).

1. Consider problem $L \in P$ and a polynomial time algorithm solving L.

Cook-Levin theorem, 1971

Theorem (Cook's theorem or Cook-Levin theorem)

SAT is NP-complete.

Direction of attack:

1. Show that every problem in NP can be solved by CircuitSAT
2. Show reduction of CircuitSAT to SAT

Lemma

Let L be a problem in P. Then there exists polynomial p and an algorithm which in time $p(n)$ builds boolean circuits B_n with n inputs and one output that solves L.
(For every $x \in \{0,1\}^*$ it holds that $B_n(x) = L(x)$.)

Proof (sketch).

1. Consider problem $L \in P$ and a polynomial time algorithm solving L.
2. For input of size n it will run in time T, use $O(T)$ cells of memory.

Cook-Levin theorem, 1971

Theorem (Cook's theorem or Cook-Levin theorem)

SAT is NP-complete.

Direction of attack:

1. Show that every problem in NP can be solved by CircuitSAT
2. Show reduction of CircuitSAT to SAT

Lemma

Let L be a problem in P. Then there exists polynomial p and an algorithm which in time $p(n)$ builds boolean circuits B_n with n inputs and one output that solves L.
(For every $x \in \{0, 1\}^*$ it holds that $B_n(x) = L(x)$.)

Proof (sketch).

1. Consider problem $L \in P$ and a polynomial time algorithm solving L.
2. For input of size n it will run in time T, use $O(T)$ cells of memory.
3. We thus need computer with memory of size $O(T)$. This can be represented by some boolean circuit.

Cook-Levin theorem, 1971

Theorem (Cook's theorem or Cook-Levin theorem)

SAT is NP-complete.

Direction of attack:

1. Show that every problem in NP can be solved by CircuitSAT
2. Show reduction of CircuitSAT to SAT

Lemma

Let L be a problem in P. Then there exists polynomial p and an algorithm which in time $p(n)$ builds boolean circuits B_n with n inputs and one output that solves L.
(For every $x \in \{0,1\}^*$ it holds that $B_n(x) = L(x)$.)

Proof (sketch).

1. Consider problem $L \in P$ and a polynomial time algorithm solving L.
2. For input of size n it will run in time T, use $O(T)$ cells of memory.
3. We thus need computer with memory of size $O(T)$. This can be represented by some boolean circuit.
4. Time of the computation can be done by using T copies of the circuit connected sequentially.

□

Cook-Levin theorem, 1971

SAT

1. Input: formula φ in CNF
2. Output: 1 if and only if there exists an $\{0, 1\}$-assignment to variables such that $\varphi(\ldots) = 1$

Cook-Levin theorem, 1971

SAT

1. Input: formula φ in CNF
2. Output: 1 if and only if there exists an $\{0,1\}$-assignment to variables such that $\varphi(\dots) = 1$

CircuitSAT

1. Input: Boolean circuit B with one output
2. Output: 1 if and only if there exists an $\{0,1\}$-assignment to inputs such that $B(\dots) = 1$

Cook-Levin theorem, 1971

SAT

1. Input: formula φ in CNF
2. Output: 1 if and only if there exists an $\{0,1\}$-assignment to variables such that $\varphi(\ldots) = 1$

CircuitSAT

1. Input: Boolean circuit B with one output
2. Output: 1 if and only if there exists an $\{0,1\}$-assignment to inputs such that $B(\ldots) = 1$

Definition (NP)

NP is the class of all (decision) problems L such that there exists some problem $K \in P$ and a polynomial g such that for every input x it holds that $L(x) = 1$ iff there exists $y \in \{0,1\}^*$ of length at most $g(|x|)$ s.t. $K(x,y) = 1$.

Cook-Levin theorem, 1971

SAT

1. Input: formula φ in CNF
2. Output: 1 if and only if there exists an $\{0,1\}$-assignment to variables such that $\varphi(\ldots) = 1$

CircuitSAT

1. Input: Boolean circuit B with one output
2. Output: 1 if and only if there exists an $\{0,1\}$-assignment to inputs such that $B(\ldots) = 1$

Definition (NP)

NP is the class of all (decision) problems L such that there exists some problem $K \in P$ and a polynomial g such that for every input x it holds that $L(x) = 1$ iff there exists $y \in \{0,1\}^*$ of length at most $g(|x|)$ s.t. $K(x,y) = 1$.

Theorem (Almost Cook-Levin thoerem)

Circuit SAT is NP-complete

Proof.

1. CircuitSAT $\in P$.

Cook-Levin theorem, 1971

SAT

1. Input: formula φ in CNF
2. Output: 1 if and only if there exists an $\{0,1\}$-assignment to variables such that $\varphi(\ldots) = 1$

CircuitSAT

1. Input: Boolean circuit B with one output
2. Output: 1 if and only if there exists an $\{0,1\}$-assignment to inputs such that $B(\ldots) = 1$

Definition (NP)

NP is the class of all (decision) problems L such that there exists some problem $K \in P$ and a polynomial g such that for every input x it holds that $L(x) = 1$ iff there exists $y \in \{0,1\}^*$ of length at most $g(|x|)$ s.t. $K(x,y) = 1$.

Theorem (Almost Cook-Levin thoerem)

Circuit SAT is NP-complete

Proof.

1. CircuitSAT $\in P$.
2. Now fix problem $L \in NP$, $K \in P$ and polynomial g. WLOG assume that $|y|$ depends only on $n = |x|$.

Cook-Levin theorem, 1971

SAT

1. Input: formula φ in CNF
2. Output: 1 if and only if there exists an $\{0,1\}$-assignment to variables such that $\varphi(\dots) = 1$

CircuitSAT

1. Input: Boolean circuit B with one output
2. Output: 1 if and only if there exists an $\{0,1\}$-assignment to inputs such that $B(\dots) = 1$

Definition (NP)

NP is the class of all (decision) problems L such that there exists some problem $K \in P$ and a polynomial g such that for every input x it holds that $L(x) = 1$ iff there exists $y \in \{0,1\}^*$ of length at most $g(|x|)$ s.t. $K(x,y) = 1$.

Theorem (Almost Cook-Levin thoerem)

Circuit SAT is NP-complete

Proof.

1. CircuitSAT $\in P$.
2. Now fix problem $L \in NP$, $K \in P$ and polynomial g. WLOG assume that $|y|$ depends only on $n = |x|$.
3. Apply lemma to obtain for given n circuit B_n solving K of size $p(g(n))$.

Cook-Levin theorem, 1971

SAT

1. Input: formula φ in CNF
2. Output: 1 if and only if there exists an $\{0,1\}$-assignment to variables such that $\varphi(\dots) = 1$

CircuitSAT

1. Input: Boolean circuit B with one output
2. Output: 1 if and only if there exists an $\{0,1\}$-assignment to inputs such that $B(\dots) = 1$

Definition (NP)

NP is the class of all (decision) problems L such that there exists some problem $K \in P$ and a polynomial g such that for every input x it holds that $L(x) = 1$ iff there exists $y \in \{0,1\}^*$ of length at most $g(|x|)$ s.t. $K(x,y) = 1$.

Theorem (Almost Cook-Levin thoerem)

Circuit SAT is NP-complete

Proof.

1. CircuitSAT $\in P$.
2. Now fix problem $L \in NP$, $K \in P$ and polynomial g. WLOG assume that $|y|$ depends only on $n = |x|$.
3. Apply lemma to obtain for given n circuit B_n solving K of size $p(g(n))$.
4. Add constants to fix input x, so the circuit only has y as an input. This is input for CircuitSAT.

Cook-Levin theorem, 1971

SAT

1. Input: formula φ in CNF
2. Output: 1 if and only if there exists an $\{0,1\}$-assignment to variables such that $\varphi(\ldots) = 1$

CircuitSAT

1. Input: Boolean circuit B with one output
2. Output: 1 if and only if there exists an $\{0,1\}$-assignment to inputs such that $B(\ldots) = 1$

Definition (NP)

NP is the class of all (decision) problems L such that there exists some problem $K \in P$ and a polynomial g such that for every input x it holds that $L(x) = 1$ iff there exists $y \in \{0,1\}^*$ of length at most $g(|x|)$ s.t. $K(x, y) = 1$.

Theorem (Almost Cook-Levin thoerem)

Circuit SAT is NP-complete

Proof.

1. CircuitSAT $\in P$.
2. Now fix problem $L \in NP$, $K \in P$ and polynomial g. WLOG assume that $|y|$ depends only on $n = |x|$.
3. Apply lemma to obtain for given n circuit B_n solving K of size $p(g(n))$.
4. Add constants to fix input x, so the circuit only has y as an input. This is input for CircuitSAT.

□

Cook-Levin theorem, 1971

Lemma

CircuitSAT can be reduced to 3-SAT

Proof.

1. Every boolean circuit can be, in polynomial time, converted to circuit only with AND and NOT gates.

Cook-Levin theorem, 1971

Lemma

CircuitSAT can be reduced to 3-SAT

Proof.

1. Every boolean circuit can be, in polynomial time, converted to circuit only with AND and NOT gates.
2. For every gate we introduce an variable representing its output.

Cook-Levin theorem, 1971

Lemma

CircuitSAT can be reduced to 3-SAT

Proof.

1. Every boolean circuit can be, in polynomial time, converted to circuit only with AND and NOT gates.
2. For every gate we introduce an variable representing its output.
3. NOT gate corresponds to CNF formula: $(x \vee y) \wedge (\neg x \vee \neg y)$.

Cook-Levin theorem, 1971

Lemma

CircuitSAT can be reduced to 3-SAT

Proof.

1. Every boolean circuit can be, in polynomial time, converted to circuit only with AND and NOT gates.
2. For every gate we introduce an variable representing its output.
3. NOT gate corresponds to CNF formula: $(x \vee y) \wedge (\neg x \vee \neg y)$.
4. AND gate corresponds to CNF formula: $(z \vee \neg x \vee \neg y) \wedge (\neg z \vee x) \wedge (\neg z \vee y)$.

Cook-Levin theorem, 1971

Lemma

CircuitSAT can be reduced to 3-SAT

Proof.

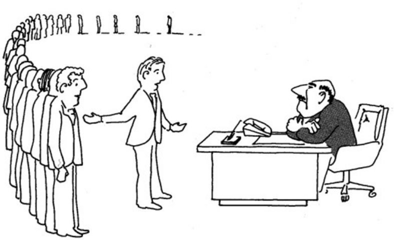
1. Every boolean circuit can be, in polynomial time, converted to circuit only with AND and NOT gates.
2. For every gate we introduce an variable representing its output.
3. NOT gate corresponds to CNF formula: $(x \vee y) \wedge (\neg x \vee \neg y)$.
4. AND gate corresponds to CNF formula: $(z \vee \neg x \vee \neg y) \wedge (\neg z \vee x) \wedge (\neg z \vee y)$.
5. Combining formulas for all gates together leads to an input to 3-SAT.

□

Proof of Cook-Levin thoerem.

We have shown that CircruitSAT is *NP*-complete and then we gave reduction to 3-SAT. We also know that 3-SAT is in NP and equivalent to SAT. □

What to do with hard problems?



"I can't find an efficient algorithm, but neither can all these famous people."

Possible ways to attack hard problems:

What to do with hard problems?

"I can't find an efficient algorithm, but neither can all these famous people."

Possible ways to attack hard problems:

1. Accept that we can solve only small inputs.

What to do with hard problems?

"I can't find an efficient algorithm, but neither can all these famous people."

Possible ways to attack hard problems:

1. Accept that we can solve only small inputs.
2. Solve a special case of the problem.

What to do with hard problems?

"I can't find an efficient algorithm, but neither can all these famous people."

Possible ways to attack hard problems:

1. Accept that we can solve only small inputs.
2. Solve a special case of the problem.
3. Find approximate solutions

What to do with hard problems?

"I can't find an efficient algorithm, but neither can all these famous people."

Possible ways to attack hard problems:

1. Accept that we can solve only small inputs.
2. Solve a special case of the problem.
3. Find approximate solutions
4. Use a heuristics

What to do with hard problems?

“I can't find an efficient algorithm, but neither can all these famous people.”

Possible ways to attack hard problems:

1. Accept that we can solve only small inputs.
2. Solve a special case of the problem.
3. Find approximate solutions
4. Use a heuristics
5. Combine above methods

Lemma

Let T be a (graph) forest and ℓ its leaf. Then at least one of the maximum independent sets in T contains ℓ.

Lemma

Let T be a (graph) forest and ℓ its leaf. Then at least one of the maximum independent sets in T contains ℓ.

IndSetInForest (T with root v, boolean array M indexed by vertices)

1. $M[v] \leftarrow$true.
2. If v is a leaf: Return.
3. For each son w of v:
4. $M \leftarrow$ IndSetInForest (subtree of T with root w, M).
5. If $m[w] =$ true: $M[v] \leftarrow$ false.
6. Return M.

Coloring interval graphs

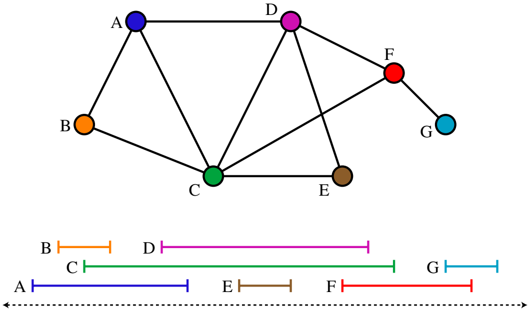
Definition (Interval graph)

An interval graph is an undirected graph formed from a set of intervals on the real line, with a vertex for each interval and an edge between vertices whose intervals intersect.

Coloring interval graphs

Definition (Interval graph)

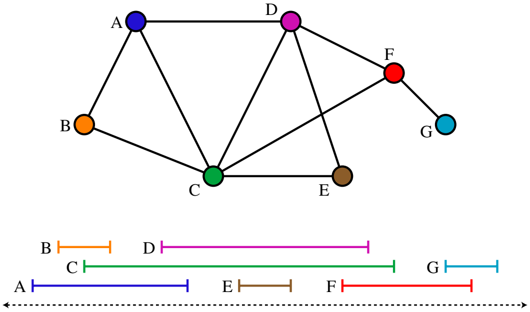
An interval graph is an undirected graph formed from a set of intervals on the real line, with a vertex for each interval and an edge between vertices whose intervals intersect.



Coloring interval graphs

Definition (Interval graph)

An interval graph is an undirected graph formed from a set of intervals on the real line, with a vertex for each interval and an edge between vertices whose intervals intersect.



IntervalGraphColoring (intervals $[x_1, y_1] \ldots [x_n, y_n]$)

$b \leftarrow 0$.
$B \leftarrow \emptyset$.
Sort set $\{x_1, y_1, \ldots, x_n, y_n\}$.
For $\{x_1, y_1, \ldots, x_n, y_n\}$ in increasing order:
    If we process some x_i:
        If $B \neq \emptyset$: Remove color from B; store it to c_i
        else: $b \leftarrow b + 1$, $c_i \leftarrow b$
    If we process some y_i:
        Add c_i to B.
Return coloring $c_1, \ldots, c_n$.

Knapsack problem

Knapsack problem

Given set of n objects with weights $w_1, \ldots, w_n$, costs $c_1, \ldots, c_n$ and maximum weight W your knapsack can carry. Find subset $P \subseteq \{1, 2, \ldots, n\}$ such that $w(P) = \sum_{i \in P} w_i$ is at most W and the cost $c(P) = \sum_{i \in P} c_i$ is maximum possible.

We can use dynamic programming to solve the problem in polynomial time in $C = \sum c_i$.

Knapsack problem

Knapsack problem

Given set of n objects with weights $w_1, \dots, w_n$, costs $c_1, \dots, c_n$ and maximum weight W your knapsack can carry. Find subset $P \subseteq \{1, 2, \dots, n\}$ such that $w(P) = \sum_{i \in P} w_i$ is at most W and the cost $c(P) = \sum_{i \in P} c_i$ is maximum possible.

We can use dynamic programming to solve the problem in polynomial time in $C = \sum c_i$.

1. Denote by $A_k(c)$ the minimum of weights of subsets $P \subseteq \{1, 2, \dots, k\}$ satisfying $c(P) = c$.

Knapsack problem

Knapsack problem

Given set of n objects with weights $w_1, \dots, w_n$, costs $c_1, \dots, c_n$ and maximum weight W your knapsack can carry. Find subset $P \subseteq \{1, 2, \dots, n\}$ such that $w(P) = \sum_{i \in P} w_i$ is at most W and the cost $c(P) = \sum_{i \in P} c_i$ is maximum possible.

We can use dynamic programming to solve the problem in polynomial time in $C = \sum c_i$.

1. Denote by $A_k(c)$ the minimum of weights of subsets $P \subseteq \{1, 2, \dots, k\}$ satisfying $c(P) = c$.
2. Proceed by induction:
   1. $A_0(0) = 0$, $A_0(1) = A_0(2) = \cdots = A_0(C) = \infty$.

Knapsack problem

Knapsack problem

Given set of n objects with weights $w_1, \dots, w_n$, costs $c_1, \dots, c_n$ and maximum weight W your knapsack can carry. Find subset $P \subseteq \{1, 2, \dots, n\}$ such that $w(P) = \sum_{i \in P} w_i$ is at most W and the cost $c(P) = \sum_{i \in P} c_i$ is maximum possible.

We can use dynamic programming to solve the problem in polynomial time in $C = \sum c_i$.

1. Denote by $A_k(c)$ the minimum of weights of subsets $P \subseteq \{1, 2, \dots, k\}$ satisfying $c(P) = c$.
2. Proceed by induction:
   1. $A_0(0) = 0$, $A_0(1) = A_0(2) = \cdots = A_0(C) = \infty$.
   2. Given A_{k-1} compute
   $$A_k(c) = \min(A_{k-1}(c), A_{k-1}(c - c_k) + w_k)$$

Knapsack problem

Knapsack problem

Given set of n objects with weights $w_1, \dots, w_n$, costs $c_1, \dots, c_n$ and maximum weight W your knapsack can carry. Find subset $P \subseteq \{1, 2, \dots, n\}$ such that $w(P) = \sum_{i \in P} w_i$ is at most W and the cost $c(P) = \sum_{i \in P} c_i$ is maximum possible.

We can use dynamic programming to solve the problem in polynomial time in $C = \sum c_i$.

1. Denote by $A_k(c)$ the minimum of weights of subsets $P \subseteq \{1, 2, \dots, k\}$ satisfying $c(P) = c$.
2. Proceed by induction:
   1. $A_0(0) = 0$, $A_0(1) = A_0(2) = \dots = A_0(C) = \infty$.
   2. Given A_{k-1} compute
   $$A_k(c) = \min(A_{k-1}(c), A_{k-1}(c - c_k) + w_k)$$
3. Once A_n is determined we know for every possible cost the subset P of that cost minimizing the weight. It remains to find maximal c such that $A_n(c) \leq W$
4. To determine the set P one can remember how the values $A_k(c)$ was determined.

(This is pseudopolynomial algorithm).