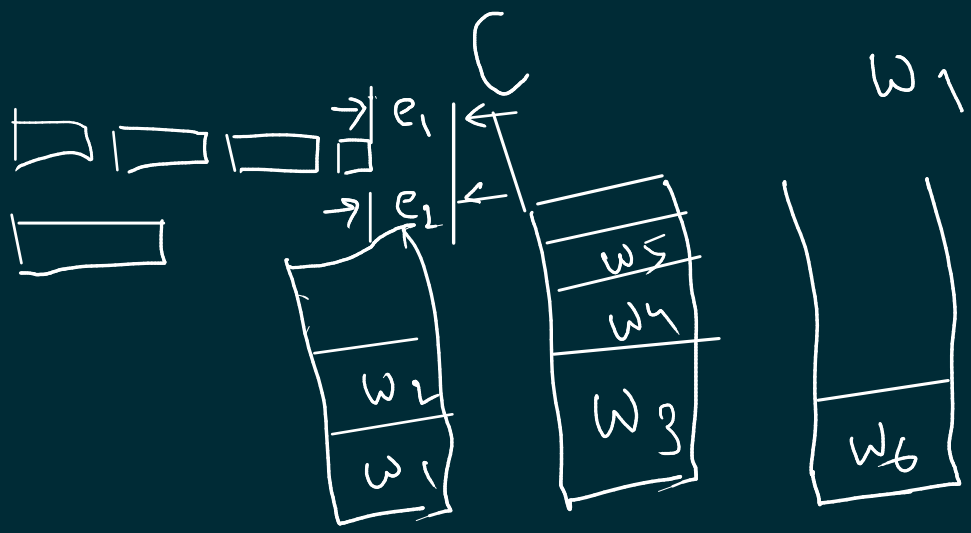
4. [Linear BIN-PACKING] The items must be packed in the bins in the order they appear in the input. Prove that the next-fit strategy solves the linear bin-packing problem exactly.

vs approximately



$w_1, w_2, \ldots, w_n$

OPT

change it to

GRD

w/o increasing # of bins

"Paragraph beautification problem"

standard "greedy" proof

$e_1^3 + e_2^3 + \cdots + e_{l-1}^3$ → DP

5. Assuming that P ≠ NP, prove that the BIN-PACKING problem cannot have a ρ-approximation algorithm for any ρ < 3/2.

(A)

PARTITION $\leq$ BIN-PACKING

$(a_1, \ldots, a_n) \longmapsto (a_1, a_2, \ldots, a_n, \frac{1}{2}\sum a_i)$

P has solution $\Rightarrow$ 2 bins suffice

$\Rightarrow$ A outputs $k < \frac{3}{2} \times 2 = 3$

$k \leq 2$

$k = 2$

P has no solution

$\Rightarrow$ OPT ≥ 3

$\Rightarrow k \geq 3$

Consider the reduction from PARTITION to BIN-PACKING in the solution of Exercise 6.22. Suppose that there exists a ρ-approximation algorithm for BIN-PACKING for $\rho < 3/2$. If two bins suffice (that is, the partitioning problem has a solution), then the algorithm will output the number of bins as $m < (3/2) \times 2 = 3$. But m is an integer, so the output will be $m = 2$. If two bins do not suffice, we have $m \geqslant 3$ anyway. Therefore the hypothetical algorithm for BIN-PACKING solves the partition problem in polynomial time, a contradiction to the NP-Completeness of PARTITION.

6. Consider the set cover problem to find the smallest cover of the set X from the collection S_1, S_2, ⋯, S_m of subsets of X. Pose this problem as a linear-programming problem.

Let X be a finite set of size n. We are given a family $S_1, S_2, \ldots, S_m$ of subsets of X with the property that $\cup_{i=1}^{m} S_i = X$. The MIN_SET_COVER problem deals with the computation of a smallest sub-collection C of the given family, which continues to cover X (that is, the union of the sets in C is X). We introduce a variable x_i for each S_i with the interpretation that

$$x_i = \begin{cases} 1 & \text{if } S_i \text{ is included in the cover } C, \\ 0 & \text{otherwise.} \end{cases}$$

The objective is to

$$\text{minimize} \sum_{i=1}^{m} x_i.$$

In order to insure that the sub-collection C covers X, take any element $a \in X$. Denote $\text{Idx}(a) = \{i \in \{1, 2, \ldots, m\} \mid a \in S_i\}$. In order that a is covered by C, we require C to contain S_i for at least one $i \in \text{Idx}(a)$. Thus, we have the following constraints:

$$\sum_{i \in \text{Idx}(a)} x_i \geqslant 1 \quad \text{for each } a \in X.$$

7. Let f denote the maximum frequency of an element of X in the subsets U_j. Relax the LP of the last question. Propose a rounding criterion to arrive at an f-approximation algorithm.

We use the ILP formulation of the MIN_SET_COVER problem discussed in Section 26.3.2. The relaxation step replaces the constraints $x_i \in \{0,1\}$ by $0 \leqslant x_i \leqslant 1$ for all $i = 1,2,\ldots,m$. Let $(x_1^*, x_2^*, \ldots, x_m^*)$ be an optimal solution of the relaxed LP instance.

Rounding this solution is not done in the literal sense. For each $a \in X$, denote by f_a the frequency of a, that is, $f_a = |\text{Idx}(a)|$. Let $f = \max_{a \in X} f_a$ denote the maximum frequency. We take the rounded solution $(x_1, x_2, \ldots, x_m)$ as

$$x_i = \begin{cases} 0 & \text{if } x_i^* < 1/f \\ 1 & \text{if } x_i^* \geqslant 1/f \end{cases}$$

for all $i = 1,2,\ldots,m$.

To prove the feasibility of the rounded solution, we take any $a \in X$. The constraint $\sum_{i \in \text{Idx(a)}} x_i^* \geqslant 1$ must be satisfied by the relaxed optimal solution. If all the x_i^* values in the sum are $< 1/f$, we have $\sum_{i \in \text{Idx(a)}} x_i^* < f_a/f \leqslant 1$, a contradiction. So $x_i^* \geqslant 1/f$ for at least one $i \in \text{Idx}(a)$. For each such i, we have taken $x_i = 1$, that is, a is covered by the rounded solution.

The rounding method ensures that $x_i \leqslant f x_i^*$ for all $i = 1,2,\ldots,n$. Therefore the approximation ratio satisfies

$$\rho \leqslant \frac{\text{SUBOPT}}{\text{ROPT}} = \frac{\sum_{i=1}^{n} x_i}{\sum_{i=1}^{n} x_i^*} \leqslant \frac{\sum_{i=1}^{n} f x_i^*}{\sum_{i=1}^{n} x_i^*} = f,$$

that is, we have an f-approximation algorithm for the minimum set cover problem.

8. An algorithm is called pseudo-polynomial-time if it runs in time polynomial in the size of the input expressed in unary. An NP-Complete problem is called weakly NP-Complete if it admits a pseudo-polynomial-time algorithm. For example, we have seen that the KNAPSACK problem is weakly NP-Complete.

Ford-Fulkerson

Prove that the SUBSET-SUM problem is weakly NP-Complete.

$a_1, a_2, \ldots, a_n, t$ Populate T

$A = a_1 + a_2 + \cdots + a_n \leq n a_{max}$ (DP)

$(t+1) \times (A+1)$ table T

$$T[i,j] = \begin{cases} 1 & \text{if } a_1, \ldots, a_i \text{ has a soln} \\ 0 & \text{if not} \end{cases}$$

Let $a_1, a_2, \ldots, a_n$ be the integers and t the target sum in an instance of SSP. Let $A = \sum_{i=1}^{n} a_i$. The unary size of this input is $O(A+n)$ (assume that $t \leqslant A$). Thus, we need to have an algorithm with running time polynomial in both n and A. Here goes one.

Build an $(n+1) \times (A+1)$ table T such that $T(i,j)$ is the decision of SSP on $a_1, a_2, \ldots, a_i, j$. The table is populated in the row-major order. The zeroth row is initialized as

$$T(0,j) = \begin{cases} 1 & \text{if } j = 0, \\ 0 & \text{if } j > 0. \end{cases}$$

Here, $i = 0$ means there are no input integers a_i, so the only sum achievable is 0.

Subsequently, for $i \geqslant 1$, we consider two cases. If $j < a_i$, then we cannot include a_i to achieve a sum of j, that is, a sum of j is achievable if and only if the first $i-1$ integers have a subcollection of sum j. On the other hand, if $j \geqslant a_i$, then we have a choice of including a_i in a subcollection. If we include a_i, we check whether the remaining sum $j - a_i$ can be achieved by a subcollection of the first $i-1$ integers. If we do not include a_i, then the sum j itself has to be achieved by a subcollection of $a_1, a_2, \ldots, a_{i-1}$. To sum up, we have

$$T(i,j) = \begin{cases} T(i-1,j) & \text{if } j < a_i, \\ T(i-1,j-a_i) \text{ OR } T(i-1,j) & \text{if } j \geqslant a_i. \end{cases}$$

Finally, we return $T(n,t)$ as the output.

This algorithm needs to compute $\leqslant (n+1)(A+1)$ entries in the table T with each entry requiring only $O(1)$ time (better $O(\log n + \log A)$ time, since i can be as large as n and j as large as A). It follows that the running time is polynomial in both n and A, as desired.

9. Prove that no NP-Complete problem about unweighted graphs can be weakly NP-Complete.

$G = (V, E)$

$|V| = n$ $\quad V = \{1, 2, \ldots, n-1\}$

$= \{1, 11, 111, \ldots, 11\ldots1\}$

$E = \{(u, v)\}$

$\rightarrow (1^u, 1^v)$

Show that the unary size of an unweighted graph is polynomially bounded by its binary size.

10. Propose an LP formulation of the BIN-PACKING problem.

$w_1, w_2, \dots, w_n, \quad C$

$$x_{ij} = \begin{cases} 1 & \text{if } w_i \text{ goes to bin } j \\ 0 & \text{otherwise} \end{cases}$$

$$\forall i \left[\sum_{j=1}^{n} x_{ij} = 1 \right] \quad i = 1, \dots, n, \quad j = 1, \dots, n$$

each item should go to a unique bin

$$\forall j \sum_{i=1}^{n} x_{ij} w_i \leq C \leftarrow \text{capacity constraint of } j\text{th bin}$$

$$y_j = \begin{cases} 1 & \text{if } j\text{-th bin is used} \\ 0 & \text{otherwise} \end{cases}$$

$$n y_j \geq \sum_{i=1}^{n} x_{ij}$$

$\min \sum y_j$

$y_1 \geq y_2 \geq y_3 \geq \dots \geq y_n$

Here is a slightly different formulation.

Clearly, n bins suffice for the bin-packing problem. We use n Boolean variables x_j, $j = 1, 2, \ldots, n$, to indicate whether the j-th bin is used. We also use n^2 variables $y_{i,j}$ to indicate whether the i-th object goes to the j-th bin. Since each object can go to exactly one bin, we have the constraints

$$\sum_{j=1}^{n} y_{i,j} = 1 \text{ for all } i = 1, 2, \ldots, n.$$

Also, the weight capacity of each bin must be respected. This gives the constraints

$$\sum_{i=1}^{n} w_i y_{i,j} \leqslant C x_j \text{ for all } j = 1, 2, \ldots, n.$$

Finally, if one wants bins with smaller numbers to fill up earlier than those with larger numbers, one may add the constraints

$$x_1 \geqslant x_2 \geqslant x_3 \geqslant \cdots \geqslant x_n.$$

The objective is to minimize $\sum_{j=1}^{n} x_j$.

11. Propose an LP formulation of the SUBSET-SUM problem.

An LP-solver not only returns the optimum objective function but also WHERE it is achieved. So even if there is no objective, the feasibility of the constraints is an issue. So the LP formulation of SSP can be as follows.

$$\text{Minimize } 0 \text{ subject to } \sum_{i=1}^{n} x_i a_i = t.$$

Here x_i is 1 if the i-th object is included in the subset; it is 0 otherwise.