1. Consider the following algorithm for the MAX-CUT problem.

A. Start with an arbitrary cut (S, T) of V.

B. So long as possible, repeat:

(i) If there exists $u \in S$ such that the cut $(S-u, T+u)$ has more cross edges than (S, T), delete u from S, and add u to T.
(ii) If there exists $v \in T$ such that the cut $(S+v, T-v)$ has more cross edges than (S, T), add v to S, and delete v from T.

C. Return (S, T).

(a) Prove that this algorithm terminates in polynomial time.

Solution: Each iteration increases the cut size by at least 1. So the number of iterations is bounded by the number of edges in G.

(b) Prove that the approximation ratio of this algorithm is 1/2.

Solution

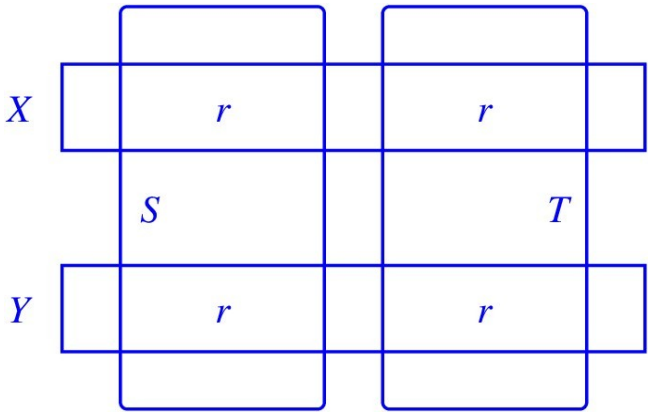
Suppose that at some point of time, S,T satisfy $|E(S,T)| < m/2$. Let c_i be the number of cross edges incident upon the i-th vertex v_i, and b_i the number of non-cross edges incident upon v_i. Clearly, $b_i + c_i = d_i$ (the degree of v_i). By the degree-sum formula,

$$2m = \sum_{i=1}^{n} d_i = \sum_{i=1}^{n} b_i + \sum_{i=1}^{n} c_i = \sum_{i=1}^{n} b_i + 2|E(S,T)| < \sum_{i=1}^{n} b_i + m,$$

that is, $\sum_{i=1}^{n} b_i > m$, that is, $\sum_{i=1}^{n} b_i > \sum_{i=1}^{n} c_i$. This implies that there must exist (at least) one vertex v_i for which $b_i > c_i$. Shifting v_i to the other part increases the number of cross edges by $b_i - c_i > 0$. To sum up, the algorithm stops after (not necessarily immediately after) $|E(S,T)| \geqslant m/2$. On the other hand, an optimal cut S^*, T^* evidently satisfies $|E(S^*,T^*)| \leqslant m$. Thus, $|E(S,T)|/|E(S^*,T^*)| \geqslant 1/2$.

(c) Prove that this approximation ratio is tight.

Solution: Consider the graph $K_{2r,2r}$, and the initial/final cut as follows.



2. Consider the following algorithm for EUCLIDEAN-TSP.

a) Select the pair (u, v) such that $c(u, v)$ is smallest among all pairs. Start with the tour $C = (u, v)$.

b) Repeat until C is a Hamiltonian cycle:

Find $i \in C$ and $j \notin C$ such that $c(i, j)$ is the minimum.
Let k be the city next to i in C.
Replace i, k by i, j, k in C.

Prove that this is a 2-approximation algorithm.

Solution

The algorithm runs for $n-2$ iterations. Let the t-th iteration replace i_t, k_t by i_t, j_t, k_t. The initial edge (u,v) and the edges (i_t, j_t) for $t = 1, 2, \ldots, n-2$ construct a minimum spanning tree of the complete weighted graph on n vertices (the cities), because Prim's algorithm chooses precisely these edges.

At the beginning, the 2-city tour has cost $d(u,v) + d(v,u) = 2d(u,v)$. The t-th iteration increases the cost of the tour by the amount $\delta_t = d(i_t, j_t) + d(j_t, k_t) - d(i_t, k_t)$. By the triangle inequality, we have $d(j_t, k_t) \leqslant d(j_t, i_t) + d(i_t, k_t)$, that is, $d(j_t, k_t) \leqslant d(i_t, j_t) + d(i_t, k_t)$, that is, $d(j_t, k_t) - d(i_t, k_t) \leqslant d(i_t, j_t)$. Therefore $\delta_t \leqslant 2d(i_t, j_t)$. The cost of the final tour produced by the algorithm is

$$c = 2d(u,v) + \sum_{t=1}^{n-2} \delta_t \leqslant 2\left[d(u,v) + \sum_{t=1}^{n-2} d(i_t, j_t)\right],$$

where the sum within square brackets is the cost of the MST.

Let OPT be the cost of the optimal tour. If we delete any edge e from the optimal tour, we get a spanning tree of cost $\text{OPT} - d(e)$. We clearly have

$$\text{OPT} \geqslant \text{OPT} - d(e) \geqslant \text{cost(MST)},$$

that is,

$$c \leqslant 2 \times \text{cost(MST)} \leqslant 2 \times \text{OPT}.$$

3. [Next-fit strategy for BIN-PACKING] Keep on adding the items to the most recently opened bin so long as possible. When the insertion of the next item lets the capacity of the current bin exceed, close the current bin, and open a new bin.

(a) Prove that this is a 2-approximation algorithm.

Solution

(a) Let m be the number of bins used by the next-fit strategy. Suppose that a bin j is at most half full in this packing. But then, the next bin $j+1$ must be more than half full. Otherwise, the opening of bin $j+1$ is not justified. More importantly, the total weight of the two bins j and $j+1$ must be larger than the bin capacity C. In fact, the sum of the weights of the items placed in bin j and the weight of the first item put in bin $j+1$ must be larger than C. It follows that among the first $m-1$ bins, at most $m/2$ bins can be at most half full, each followed by a bin more than half full that makes the average weight of the two consecutive bins larger than $C/2$. Therefore the total weight packed is $\sum_{i=1}^{n} a_i > Cm/2$. Moreover, $C \times \text{OPT} \geqslant \sum_{i=1}^{n} a_i$, that is, $m < 2 \times \text{OPT}$. Since m and OPT are integers, we have $m \leqslant 2 \times \text{OPT} - 1$.

(b) Prove that the approximation ratio of 2 is tight, that is, given any $\varepsilon > 0$, there exists a collection for which the approximation ratio is $> 2 - \varepsilon$.

Solution

(b) Given $\varepsilon > 0$, choose a positive integer $k > \frac{1}{\varepsilon}$. Let there be $k-1$ items of weight k, and k items of weight 1, and $C = k$. Suppose that these items occur in the sequence $1, k, 1, k, \ldots, 1, k, 1$. We have $\text{OPT} = k$ (place all the objects of weight 1 in one bin, and each item of weight k in a single bin; this strategy must be optimal since all the k bins are filled to the capacity). The next-fit algorithm uses $m = 2k - 1$ bins. The approximation ratio is therefore $2 - \frac{1}{k} > 2 - \varepsilon$.

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

Solution

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.

5. 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.

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

Solution

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.

6. Let A = (a_1, a_2, . . . , a_n) be an array of n positive integers, and t a target sum (a positive integer again). The task is to find a subset $I \subseteq \{1, 2, 3, \ldots, n\}$ for which the sum of a_i for $i \in I$ is as small as possible but at least as large as t. Assume that $t \leq a_1 + a_2 + \cdots + a_n$ (otherwise the problem has no solution). Prof. Sad proposes the following algorithm to solve this problem.

Initialize sum = 0, and $I = 0$.
for $i = 1, 2, 3, \ldots,$ n (in that order), repeat {
  Update sum = sum + a_i , and $I = I \cup \{i\}$.
  If sum $\geq t$, break.
}
Return I.

(a) Prove that the approximation ratio of Prof. Sad's algorithm cannot be restricted by any constant value.

Solution Take any constant $\Delta \geqslant 1$ (may be very large). Consider the instance $n = 2$, $A = (\Delta + 1, 1)$, and $t = 1$. Then, Prof. Sad's algorithm returns the index set $\{1\}$ of sum $\Delta + 1$, whereas the optimal solution is $\{2\}$ of sum 1. So the approximation ratio is $\Delta + 1 > \Delta$.

(b) Prof. Atpug suggests sorting the array A in the ascending order before running the algorithm. In view of this suggestion, we now have $a_1 \leq a_2 \leq a_3 \leq \cdots \leq a_n$. Prove that Prof. Atpug's suggestion makes Prof. Sad's algorithm a 2-approximation algorithm.

Solution We have $I = \{1, 2, 3, \ldots, k\}$ for some $k \geqslant 1$ satisfying $a_1 + a_2 + \cdots + a_{k-1} < t$ but $a_1 + a_2 + \cdots + a_{k-1} + a_k \geqslant t$. Now, consider the two cases.

Case 1: $a_k < t$. Here, $a_1 + a_2 + \cdots + a_{k-1} + a_k = (a_1 + a_2 + \cdots + a_{k-1}) + a_k < t + t = 2t$, whereas the optimal sum satisfies OPT $\geqslant t$.

Case 2: $a_k \geqslant t$. Here, the optimal sum is OPT $= a_k$ (because the preceeding numbers $a_1, a_2, \ldots, a_{k-1}$ (if any) add up to a sum smaller than t, whereas $a_{k+1}, a_{k+2}, \ldots, a_n$ are each at least as large as a_k). But then, $a_1 + a_2 + \cdots + a_{k-1} < t \leqslant a_k$, whereas the algorithm gives the sum $(a_1 + a_2 + \cdots + a_{k-1}) + a_k < a_k + a_k = 2a_k$.

7.

You are given a stream of songs lasting for $t_1, t_2, t_3, \ldots, t_n$ seconds. You have two write-once devices D_1 and D_2, each capable of storing songs of total duration T seconds. When the i-th song starts, you have three options: (i) copy the song to D_1, (ii) copy the song to D_2, and (iii) discard the song. The copy of a song to a device is allowed only when there is enough memory left in that device. Assume that the individual song durations t_i are known to you beforehand, and these durations satisfy $t_i \leqslant T$ for all i, and $\sum_{i=1}^{n} t_i \leqslant 2T$. Your goal is to maximize the total copy time in the two output devices together. To that effect, you run the following greedy algorithm. Derive a tight approximation ratio of the algorithm. Prove that the approximation ratio is tight.

```
for i = 1,2,3,...,n (in that order) {
    If D1 can accommodate the i-th song, copy the i-th song to D1,
    else if D2 can accommodate the i-th song, copy the i-th song to D2,
    else discard the i-th song, and continue.
}
```

Solution

If all the songs can be copied to D_1 and D_2, we have obtained an optimal solution. So assume that some song(s) can be copied to neither D_1 nor D_2. Let the k-th song be the first such song. Under the assumption $t_k \leqslant T$, this implies that both D_1 and D_2 store some songs at the time when the k-th song comes. This implies that the first $k-1$ songs could not be copied to D_1 alone because of its capacity constraint, that is, the total length of the songs copied so far is already $> T$. Since the two devices can store a maximum of $2T$ seconds of songs, the approximation ratio is $> \frac{1}{2}$.

In order to see that this approximation ratio is tight, start with any $\varepsilon > 0$. Take $T \geqslant \frac{1}{\varepsilon}$. Consider the four songs of durations $1, \frac{T}{2}, \frac{T}{2}, T-1$. The greedy algorithm copies the first two songs to D_1 and the third song to D_2, and discards the last song. All the songs can be copied to the devices (we have $1+(T-1) = \frac{T}{2} + \frac{T}{2} = T$), so the approximation ratio is $\frac{T+1}{2T} = \frac{1}{2} + \frac{1}{T} \leqslant \frac{1}{2} + \varepsilon$.

8. Let $G = (V, E)$ be a connected undirected graph. You make a DFS traversal of G starting from any arbitrary vertex. Let T be the DFS tree produced by the traversal. Take C to be the set of all internal (that is, non-leaf) nodes in T.

(a) Prove that C is a vertex cover for G.

(b) Prove that determining C using this method is a 2-approximation algorithm for the MIN-VERTEX-COVER problem.

Solution

(a) Let $e = (u, v) \in E$. Then e is either a tree edge (an edge of T) or a back edge. In both the cases, at least one of u and v must be a non-leaf node in T, that is, e is covered by C. (This is not true for a BFS tree, because the graph may contain cross edges connecting pairs of leaf nodes.)

(b) There exists a matching M in T (and so in G) in which every internal vertex of T is matched (this is true for any tree, not necessarily a DFS tree). In order to see why, let us start at the root r with $M = \emptyset$. If r is a leaf node, we are done. So suppose that $u_1, u_2, \ldots, u_k$ are the children of r. Match r to u_1 (that is, add (r, u_1) to M). Then, recursively carry out the procedure on $u_2, u_3, \ldots, u_k$ and on the children of u_1 (if any).

Let t be the number of internal nodes in T, that is, $|C| = t$. Since each edge of M has an internal vertex as at least one endpoint, we have $t \leqslant 2|M|$. Moreover, $\text{OPT} \geqslant |M|$ (any vertex cover is of size at least that of any matching), so $t \leqslant 2 \times \text{OPT}$.

9. Consider the knapsack problem. First, sort the objects in decreasing order of p_i / w_i . Call this sorted list $O_1, O_2, \ldots, O_n$. Assume that $w_1 + w_2 + \cdots + w_n > C$ (otherwise the solution is trivial). Let k be the index such that $w_1 + w_2 + \cdots + w_k \leq C$, but $w_1 + w_2 + \cdots + w_{k+1} > C$. Now, let j be the index such that p_j is maximum. Augment the greedy algorithm to output $\{1, 2, \ldots, k\}$ or $\{ j \}$, whichever gives better profit. Prove that this is a 1/2-approximation algorithm for the knapsack problem.

Solution

For a moment, consider the optimal solution FOPT of the fractional knapsack problem. Let $\alpha = (C - \sum_{i=1}^{k} w_i)/w_{k+1}$. Then,

$$\text{FOPT} = \left(\sum_{i=1}^{k} p_i \right) + \alpha p_{k+1}.$$

Since the 0,1 knapsack problem is not allowed to pack fractions of items, its optimal profit OPT must satisfy

$$\text{OPT} \leqslant \text{FOPT}.$$

Combining these two observations, we have

$$\text{OPT} \leqslant \left(\sum_{i=1}^{k} p_i \right) + \alpha p_{k+1}.$$

This, in turn, indicates that either $\sum_{i=1}^{k} p_i \geqslant \frac{1}{2}\text{OPT}$ or $\alpha p_{k+1} \geqslant \frac{1}{2}\text{OPT}$. The second inequality implies

$$p_j \geqslant p_{k+1} \geqslant \frac{\frac{1}{2}\text{OPT}}{\alpha} \geqslant \frac{1}{2}\text{OPT},$$

since α is a fraction less than 1.