Chapter 3: The Greedy Method

1. (a) Describe an $O(n \log n)$-time algorithm that, given a set S of n integers and another integer x, determines whether or not there exist two elements in S whose sum is exactly x. (b) Please give an example to illustrate your algorithm.

   Sol.

   (a) The following algorithm solves the problem:

   Step 1. Sort the elements in S.

   Step 2. Form the set $\hat{S} = \{z \geq 0 \mid z = x - y \text{ for } y \in S\}$.

   Step 3. Sort the elements in $\hat{S}$.

   Step 4. If any value in S appears more than once, remove all but one instance. Do the same for $\hat{S}$.

   Step 5. Merge the two sorted sets S and $\hat{S}$ by calling Algorithm Liner-Merge.

   Step 6. There exist two elements in S whose sum is exactly x if and only if the same value appears in consecutive positions in the merged list.

   Steps 1 and 3 require $O(n \log n)$ steps. Steps 2, 4, 5, and 6 require $O(n)$ steps. Thus the overall running time is $O(n \log n)$.

   [The correctness of the above algorithm is as follows. To justify the claim in step 4, first observe that if any value appears twice in the merged output, it must appear in consecutive positions. Thus, we can restate the condition in step 5 as there exist two elements in S whose sum is exactly x if and only if the same value appears twice in the merged output.

   Suppose that some value w appears twice. Then w appeared once in S and once in $\hat{S}$. Because w appeared in $\hat{S}$, there exists some $y \in S$ such that $w = x - y$, or $x = w + y$. Since $w \in S$, the elements w and y are in S and sum to x.

   Conversely, suppose that there are values $w, y \in S$ such that $w + y = x$. Then, since $x - y = w$, the value w appears in $\hat{S}$. Thus, w is in both S and $\hat{S}$, and so it will appear twice in the merged output.]

   (b) Example: $S = 2, 13, 19, 5$, and $x = 15$.

   First, sort S in an increasing manner. Then, $S = 2, 5, 13, 19$.

   Second, construct $\hat{S} = 13, 10, 2$ and sort it. Then, $\hat{S} = 2, 10, 13$.

   Third, remove all the same elements in S and $\hat{S}$, but reserve one in the list.

   Finally, merge S and $\hat{S}$ into a sorted list $L = \underline{2, 2}, 5, 10, \underline{13, 13}, 19$.

   If there exist the same elements in the merged list L, then they are the answer. For example, in L 2 and 13 are the same values appeared in L, and hence they are the answer.

2. What are the minimum and maximum numbers of elements in a heap of height

h?

Sol.

Since a heap is an almost-complete binary tree (complete at all levels except possibly the lowest), it has at most $\mathbf{2^{h+1}-1}$ elements (if it is complete) and at least $2^h-1+1 = \mathbf{2^h}$ elements (if the lowest level has just 1 element and the other levels are complete).

3. Show that an n-element heap has height $\lfloor \log n \rfloor$?

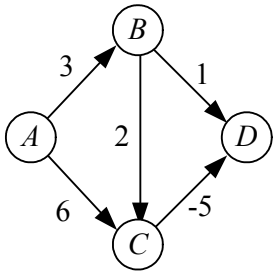
Sol.

Given an n-element heap of height h, we know that $2^h \le n \le 2^{h+1}-1 < 2^{h+1}$.
Thus, $h \le \log n < h+1$. Since h is an integer, $h = \lfloor \log n \rfloor$ (by definition of $\lfloor\ \rfloor$).

4. (a) Show why Dijkstra's algorithm will not work properly when the considered graph contains negative cost edges. (b) Modify Dijkstra's algorithm so that it can compute the shortest path from source node to each node in an arbitrary graph with negative cost edges, but no negative cycles.

Sol.

(a) Consider the following graph.



If the Dijkstra's algorithm is used, we would have the following result:

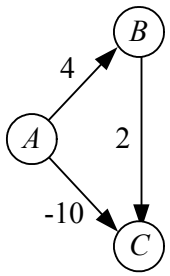
A	0
AB	3
ABD	4

Thus, we would conclude that the shortest path from A to D is through B and the length is 4. Actually, the shortest path is $A \to B \to C \to D$ whose length is $3+2+(-5) = 0$.

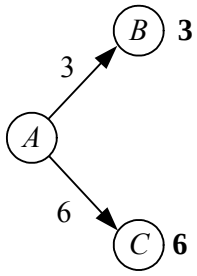
Dijkstra's algorithm does not work in this case due to the existence of negative weights. Dijkstra's algorithm is a greedy algorithm. When a node u is considered, all of the nodes which are directly linked to u are considered and the shortest one is selected. Suppose v is selected. Then the shortest distance from the origin to v is now found. With negative weights, this mechanism is no longer valid as shown in the above example.

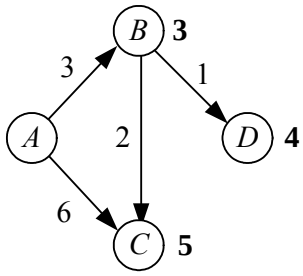
(b) The Bellman and Ford algorithm can be used in this case if there is no

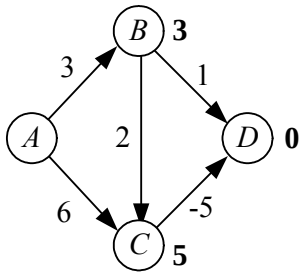
negative cycle, that is, a cycle whose total cost is negative, as shown in the following figure:



The following is an illustration of the Bellman and Ford algorithm to find all shortest paths. As can be seen, if there are n nodes, there will be $n-1$ passes. In each pass, we update the shortest distance from the origin node to a node.







5. 【Node Cover】Let $G = (V, E)$ be an undirected graph. A node cover of G is a subset U of the vertex set V such that every edge in E is incident to at least one vertex in U. A minimum node cover is one with the fewest number of vertices. Consider the following greedy algorithm for this problem: (the degree of a vertex

is the number of vertices adjacent to it)

Algorithm Cover(int V[], int E[])

{

$U = \phi$;

Do {

Let q be a vertex from V of maximum degree;

Add q to U; eliminate q from V;

$E = E - \{(x, y) \mid x = q \text{ or } y = q\}$;

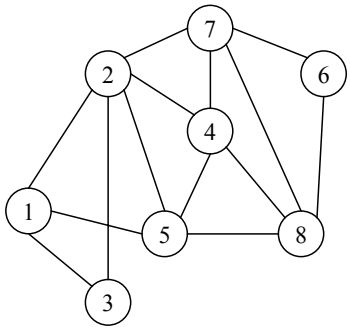
} while ($E \neq \phi$)

Output U; // U is the node cover

}

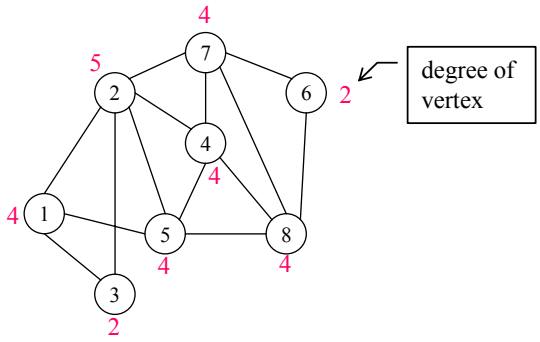
(a) Please use the above algorithm to find a node cover of the following graph?

(b) Does the above algorithm always generate a minimum node cover?

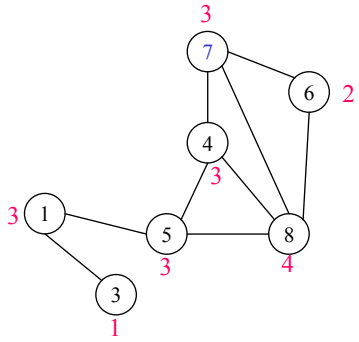


Sol.

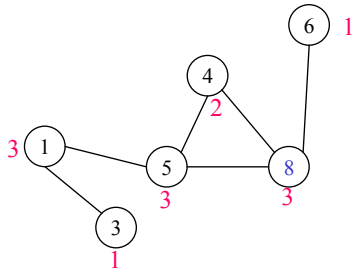
(a)



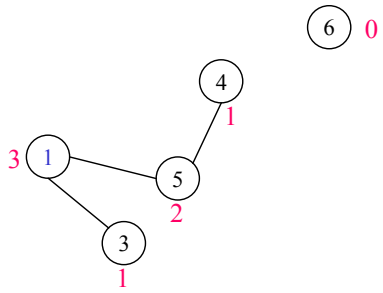
$U = \{2\}$ since vertex 2 has the maximum degree.



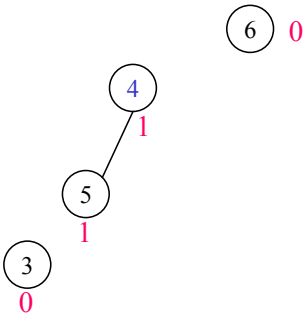
$U = \{2, 7\}$.



$U = \{2, 7, 8\}$.

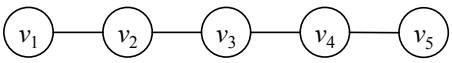


$U = \{2, 7, 8, 1\}$.



$U = \{2, 7, 8, 1, 4\}$: This is the output of Algorithm Cover.

(b) No. Give a counter-example: For example, Algorithm Cover may output $U = \{v_3, v_2, v_4\}$. But, the minimum node cover is $\{v_2, v_4\}$. Thus, Algorithm Cover does not always output the minimum node cover. Note that Algorithm Cover may produce an optimal solution for the following graph.



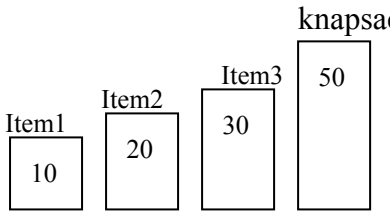
6. 【0/1 Knapsack Problem】(a) What is the 0/1 Knapsack Problem? (b) Please give a greedy algorithm, which is heuristic, to solve the 0/1 Knapsack Problem. (Note that the solution of your algorithm may not be optimal) (c) Please give an example to show that it does not always yield an optimal solution.

Sol.

(a) Given positive integers $p_1, p_2, p_3, \ldots, p_n, w_1, w_2, w_3, \ldots, p_n$, and M. The problem is to find $x_1, x_2, x_3, \ldots, x_n$, $\boldsymbol{x_i = 0 \text{ or } 1}$, such that $\sum_{i=1}^{n} p_i x_i$ is maximized subject to $\sum_{i=1}^{n} w_i x_i \le M$.

(b) A heuristic algorithm orders the items according to v_i / w_i in the ascending way. We select the items by choosing the item with the largest v_i / w_i, the second largest v_i / w_i until the capacity of the knapsack is exceeded. This is a typical greedy algorithm.

(c) A counterexample is illustrated as below:



The value of v_i / w_i :

Item 1 60/10 = 6

2 100/20 = 5

3 120/30 = 4

Our greedy algorithm will select items 1 and 2 whose total value is 160. The optimum solution is actually to choose items 2 and 3 whose total value is 220.

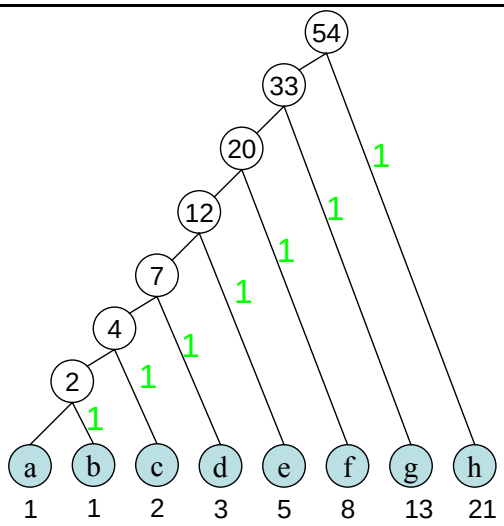
7. (a) What is an optimal Huffman code for the following set of frequencies, based on the first 8 Fibonacci numbers?

a: 1 b:1: c:2 d:3 e:5 f:8 g:13 h:21

(b) Can you generalize your answer to find the optimal code when the frequencies are the first *n* Fibonacci numbers?

Sol.

(a) The Huffman tree is as follows:



Huffman codes⇒ h: 1; g:01; f:001; e:0001; d:00001; c:000001; b:0000001; a:0000000.

(b) Generalization:

*n*th number: 1

(*n*−1)th number: 01

(*n*–*j*)th number: 00...001 (*j* 0's)

...

1st number: 00...000 (*n*−1 0's)

Thus, the Huffman code is as follows:

$$j\text{th number}: \begin{cases} \underbrace{00\cdots00}_{n-1} & \text{, if } j=1; \\ \underbrace{00\cdots00}_{n-j}1 & \text{, if } 2 \le j \le n. \end{cases}$$