COMP 355
Advanced Algorithms

Subset-Sum

Sections 6.4 & 8.8(KT)

Section 35.5(CLRS)

Rhodes College

1

Subset Sum Problem

Subset Sum (SS): Given a finite set S of positive integers S = {w_1,w_2, . . . ,w_n} and a target value, t, we want to know whether there exists a subset S′ ⊆ S that sums **exactly** to t.

Example:

S = {3, 6, 9, 12, 15, 23, 32} and t = 33.

The subset S' = {6, 12, 15} sums to t = 33, so the answer in this case is yes. If t = 34, the answer would be no.

2

Recall 0-1 Knapsack

0-1 Knapsack Problem: Given a collection of objects, each with an associated weight w_i and associated value v_i, and a knapsack capacity W, fill the knapsack in such a way as to maximize the value of the objects without exceeding W (capacity).

Simplest version: Suppose that the value is the same as the weight, $v_i = w_i$. (This would occur for example if all the objects were made of the same material, say, gold.)

Then, the best we could hope to achieve would be to fill the knapsack entirely. (Set t = W) (equivalent to Subset Sum)

3

Dynamic Programming Solution

- There is a dynamic programming algorithm which solves the Subset Sum problem in $O(n \cdot t)$ time.
- The quantity $n \cdot t$ is a polynomial function of n.
  - Seems to imply what?

Recall that in all NP-complete problems we assume:

1. Running time is measured as a function of input size (number of bits)
2. Inputs must be encoded in a reasonable succinct manner

- Let us assume that the numbers w_i and t are all b-bit numbers represented in base 2, using the fewest number of bits possible.
- Then the input size is O(nb). The value of t may be as large as 2^b.
- Resulting algorithm has a running time of $O(n2^b)$. (polynomial in n, but exponential in b.)
- We will show that in the general case, this problem is NP-complete.

4

SS is NP-complete

The proof that Subset Sum (SS) is NP-complete involves the usual two elements.

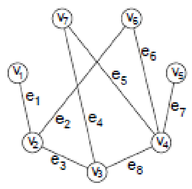
i. SS ∈ NP.

ii. Some known NP-complete problem is reducible to SS. In particular, we will show that Vertex Cover (VC) is reducible to SS, that is, VC $\leq_P$ SS.

SS ∈ NP. Given S and t, the certificate is just the indices of the numbers that form the subset S'. We can add two b-bit numbers together in O(b) time. So, in polynomial time we can compute the sum of elements in S', and verify that this sum equals t.

An Initial Approach

Here is an idea, which **does not work**, but gives a sense of how to proceed.

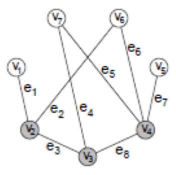


$$\begin{array}{c|cccccccc|}
 & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 \\
\hline
v_1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
v_2 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
v_3 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\
v_4 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
v_5 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
v_6 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
v_7 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
\hline
\end{array}$$

Encoding a graph as a collection of bit vectors.

Let E denote the number of edges in the graph.

1. Number the edges of the graph from 1 through E.
2. Represent each vertex v_i as an E-element bit vector, where the j-th bit from the left is set to 1 if and only if the edge e_j is incident to vertex v_i.



$$\begin{array}{c|cccccccc|l}
 & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 & \\
\hline
v_1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
v_2 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & \\
v_3 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & \\
v_4 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & \\
v_5 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \\
v_6 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & \\
v_7 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & \\
\hline
t = & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & v_2 \vee v_3 \vee v_4
\end{array}$$

The logical-or of a vertex cover equals 1111 . . . 1.

Take any subset of vertices and form the logical-or of the corresponding bit vectors.

- If the subset is a vertex cover, every edge will be covered by at least one of these vertices, (logical-or will be a bit vector of all 1's, 1111 . . . 1)
- Conversely, if the logical-or is a bit vector of 1's, then each edge has been covered by some vertex, implying that the vertices form a vertex cover

The Final Reduction

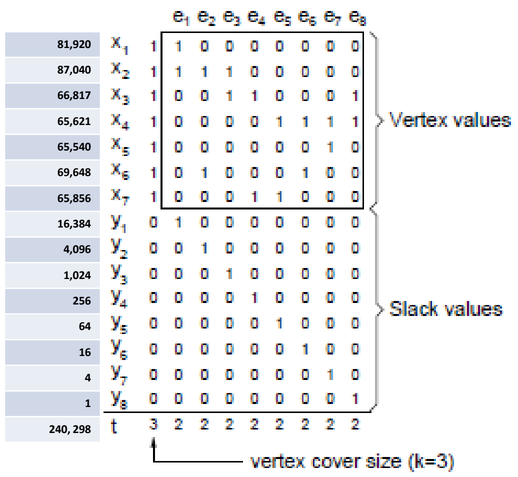
Given the graph G = (V, E) and integer k for the vertex cover problem.

1. Create a set of n vertex values, $x_1, x_2, \ldots, x_n$ using base-4 notation. The value x_i is equal to a 1 followed by a sequence of E base-4 digits. The j-th digit is a 1 if edge e_j is incident to vertex v_i and 0 otherwise.
2. Create E slack values $y_1, y_2, \ldots, y_E$, where y_i is a 0 followed by E base-4 digits. The i-th digit of y_i is 1 and all others are 0.
3. Let t be the base-4 number whose first digit is k **(this may actually span multiple base-4 digits),** and whose remaining E digits are all 2.
4. Convert the x_i's, the y_j 's, and t into whatever base notation is used for the subset sum problem (e.g. base 10). Output the set $S = \{x_1, \ldots, x_n, y_1, \ldots, y_E\}$ and t.

Observe that this can be done in polynomial time, in $O(E^2)$, in fact.

7

VC to SS Reduction



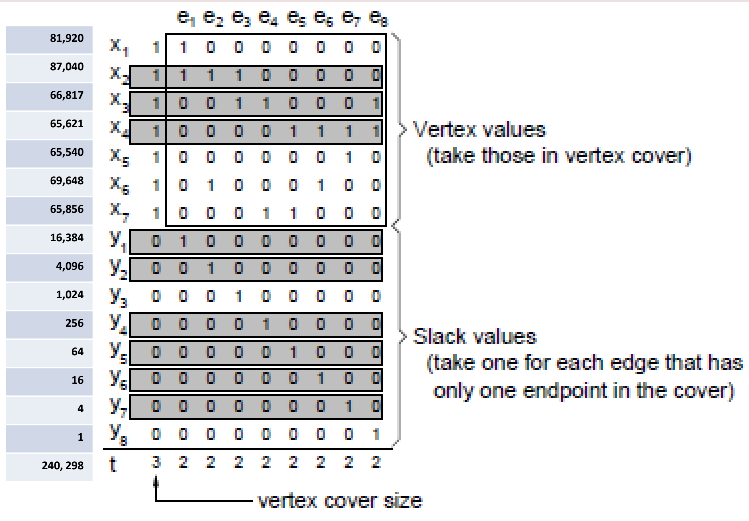
			e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
81,920	x_1	1	1	0	0	0	0	0	0	0
87,040	x_2	1	1	1	1	0	0	0	0	0
66,817	x_3	1	0	0	1	1	0	0	0	1
65,621	x_4	1	0	0	0	0	1	1	1	1
65,540	x_5	1	0	0	0	0	0	0	1	0
69,648	x_6	1	0	1	0	0	0	1	0	0
65,856	x_7	1	0	0	0	1	1	0	0	0
16,384	y_1	0	1	0	0	0	0	0	0	0
4,096	y_2	0	0	1	0	0	0	0	0	0
1,024	y_3	0	0	0	1	0	0	0	0	0
256	y_4	0	0	0	0	1	0	0	0	0
64	y_5	0	0	0	0	0	1	0	0	0
16	y_6	0	0	0	0	0	0	1	0	0
4	y_7	0	0	0	0	0	0	0	1	0
1	y_8	0	0	0	0	0	0	0	0	1
240, 298	t	3	2	2	2	2	2	2	2	2

Vertex values
Slack values
vertex cover size (k=3)

Vertex cover to subset sum reduction.

8

Correctness of the Reduction



			e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
81,920	x_1	1	1	0	0	0	0	0	0	0
87,040	x_2	1	1	1	1	0	0	0	0	0
66,817	x_3	1	0	0	1	1	0	0	0	1
65,621	x_4	1	0	0	0	0	1	1	1	1
65,540	x_5	1	0	0	0	0	0	0	1	0
69,648	x_6	1	0	1	0	0	0	1	0	0
65,856	x_7	1	0	0	0	1	1	0	0	0
16,384	y_1	0	1	0	0	0	0	0	0	0
4,096	y_2	0	0	1	0	0	0	0	0	0
1,024	y_3	0	0	0	1	0	0	0	0	0
256	y_4	0	0	0	0	1	0	0	0	0
64	y_5	0	0	0	0	0	1	0	0	0
16	y_6	0	0	0	0	0	0	1	0	0
4	y_7	0	0	0	0	0	0	0	1	0
1	y_8	0	0	0	0	0	0	0	0	1
240, 298	t	3	2	2	2	2	2	2	2	2

Vertex values
(take those in vertex cover)
Slack values
(take one for each edge that has only one endpoint in the cover)
vertex cover size

In our dynamic programming solution W = t, so the DP algorithm would run in $\Omega(n4^n)$ time, which is not polynomial time.

Correctness

We claim that G has a vertex cover of size k iff S has a subset that sums to t.

$\Rightarrow$ If G has a vertex cover V′ of size k, then we take the vertex values x_i corresponding to the vertices of V′, and for each edge that is covered only once in V′, we take the corresponding slack variable. It follows from the comments made earlier that the lower-order E digits of the resulting sum will be of the form 222 . . . 2 and because there are k elements in V′, the leftmost digit of the sum will be k. Thus, the resulting subset sums to t.

$\Leftarrow$ If S has a subset S′ that sums to t then we assert that it must select exactly k values from among the vertex values, since the first digit must sum to k. We claim that these vertices V′ form a vertex cover. In particular, no edge can be left uncovered by V′, since (because there are no carries) the corresponding column would be 0 in the sum of vertex values. Thus, no matter what slack values we add, the resulting digit position could not be equal to 2, and so this cannot be a solution to the subset sum problem.

Polynomial Approximation Schemes

To control the precision of the approximation.

1. Specify a parameter $\epsilon > 0$ as part of the input to the approximation algorithm
2. Require that the algorithm produce an answer that is within a relative error of ϵ of the optimal solution.

   Note: It is understood that as ϵ tends to 0, the running time of the algorithm will increase.

Such an algorithm is called a polynomial approximation scheme.

Ex: Running time = $O(2^{(1/\epsilon)}n^2)$

A fully polynomial approximation scheme is one in which the running time is polynomial in both n and $1/\epsilon$.

Ex. Running time = $O((n/\epsilon)^2)$