GREEDY ALGORITHMS

Change-Making Problem

- Assume that you have the coins below.

States are $d_1 = 25$ (quarter), $d_2 = 10$ (dime), $d_3 = 5$ (nickel), and $d_4 = 1$ (penny).

- How would you give change with coins of these denominations of, say, 48 cents?

Change-Making Problem

- If you came up with the answer 1 quarter, 2 dimes, and 3 pennies, you followed—consciously or not—a logical strategy of making a sequence of best choices among the currently available alternatives. Indeed, in the first step, you could have given one coin of any of the four denominations.
- “Greedy” thinking leads to giving one quarter because it reduces the remaining amount the most, namely, to 23 cents.
- In the second step, you had the same coins at your disposal, but you could not give a quarter, because it would have violated the problem’s constraints.
- So your best selection in this step was one dime, reducing the remaining amount to 13 cents. Giving one more dime left you with 3 cents to be given with three pennies.

What is greedy?

- The approach applied in the opening paragraph to the change-making problem is called greedy.
- Computer scientists consider it a general design technique despite the fact that it is applicable to optimization problems only.
- The greedy approach suggests constructing a solution through a sequence of steps, each expanding a partially constructed solution obtained so far, until a complete solution to the problem is reached. On each step—and this is the central point of this technique—the choice made must be:

What is greedy?

- feasible, i.e., it has to satisfy the problem’s constraints
- locally optimal, i.e., it has to be the best local choice among all feasible choices available on that step
- irrevocable, i.e., once made, it cannot be changed on subsequent steps of the algorithm

What is greedy?

- These requirements explain the technique’s name: on each step, it suggests a “greedy” grab of the best alternative available in the hope that a sequence of locally optimal choices will yield a (globally) optimal solution to the entire problem.
- We refrain from a philosophical discussion of whether greed is good or bad. (If you have not seen the movie from which the chapter’s epigraph is taken, its hero did not end up well.)
- From our algorithmic perspective, the question is whether such a greedy strategy works or not.
- As we shall see, there are problems for which a sequence of locally optimal choices does yield an optimal solution for every instance of the problem in question.
- However, there are others for which this is not the case; for such problems, a greedy algorithm can still be of value if we are interested in or have to be satisfied with an approximate solution.

Change-Making Problem

- If you have 20, 19, 5, 1 coins and you try to have 24 how can you do that?
- Greedy algorithms always try to approach the answer mostly so it firstly will choose 20 coin.
- After that 4 remaining and the algorithms complete these 4 with 4 pennies.
- So it becomes totally 5 coins.
- 20 + 1 + 1 + 1 + 1

Change-Making Problem

- But there is a better colution such as 19 + 5
- It becomes only two coins.

Change-Making Problem

- If you have 10, 9 and 1 coins and you want to pay 37 cents, what are the solutions?
- Greedy solutions choose the most valuable coin first of all. It is 10 cent.
- We pay 1 10 cent and 37-10 = 27 is remaining
- Greedy algorithm choose again 10 for paying 27

- So we pay 2 10 cent and 37 – 10 – 10 = 17 is remaining
- Greedy algorithm choose again 10 for paying 17

Change-Making Problem

- So we pay 3 10 cent and 37 – 10 – 10 – 10 = 7 is remaining
- Greedy algorithm choose 1 cent for paying 7
- We pay 7 1 cents for remaining 7.

- So the greedy algorithm solution is such as:
- 10 + 10 + 10 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 37
- There are 10 coins.

Change-Making Problem

- Is there a more efficient solution?
- Yes there is.
- 10 + 9 + 9 + 9 = 37
- You can pay 37 cents with only 4 coins.

Change-Making Problem

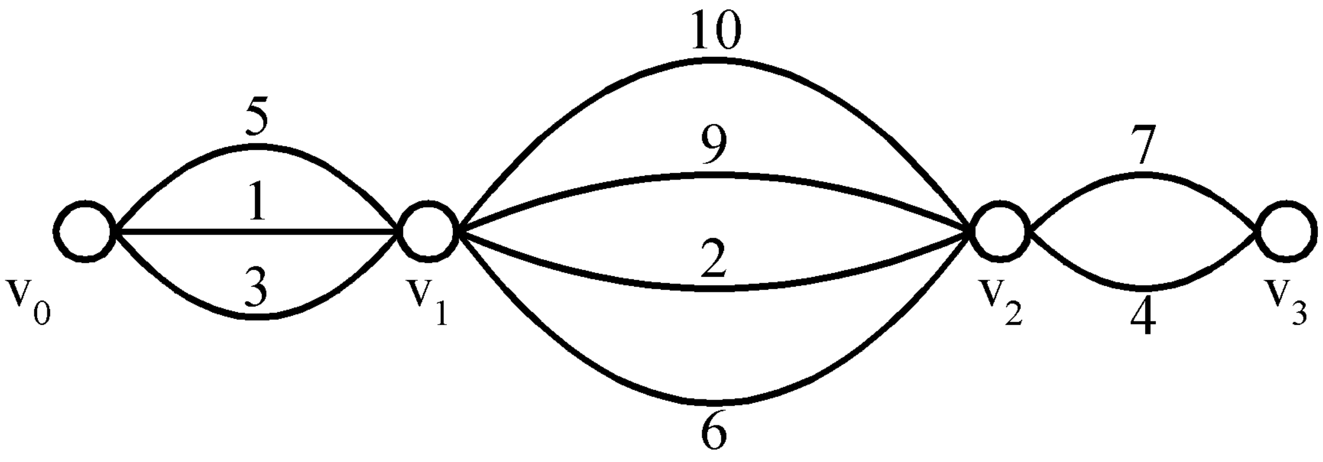
- Another question?
- If you have coins for 1, 4, 5 and 10 cent coins and you need to pay 8 cents, how can you pay?

- First of all, Greedy algorithm chooses the mos valuable coin which is less then 8.
- It is 5.
- So 8 – 5 = 3 remaining.
- The algorithm makes 3 1 cents for 3 cents.

Change-Making Problem

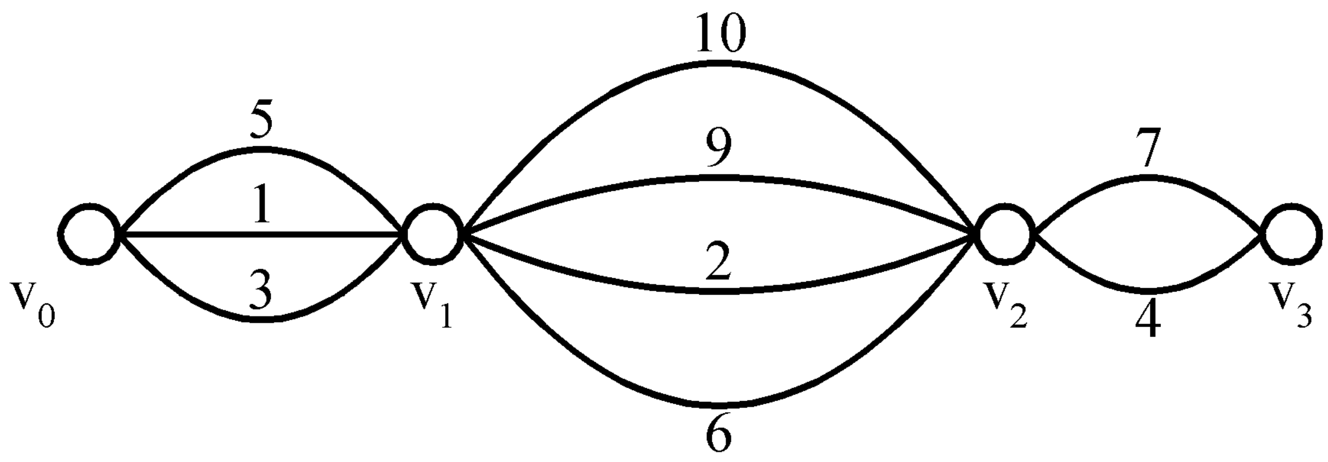
- So the solution is
- $5 + 1 + 1 + 1 = 8$
- The solution has 4 coins.
- But if we write $4 + 4 = 8$ cents, the problem is solved with only 2 coins.

Graph Problems

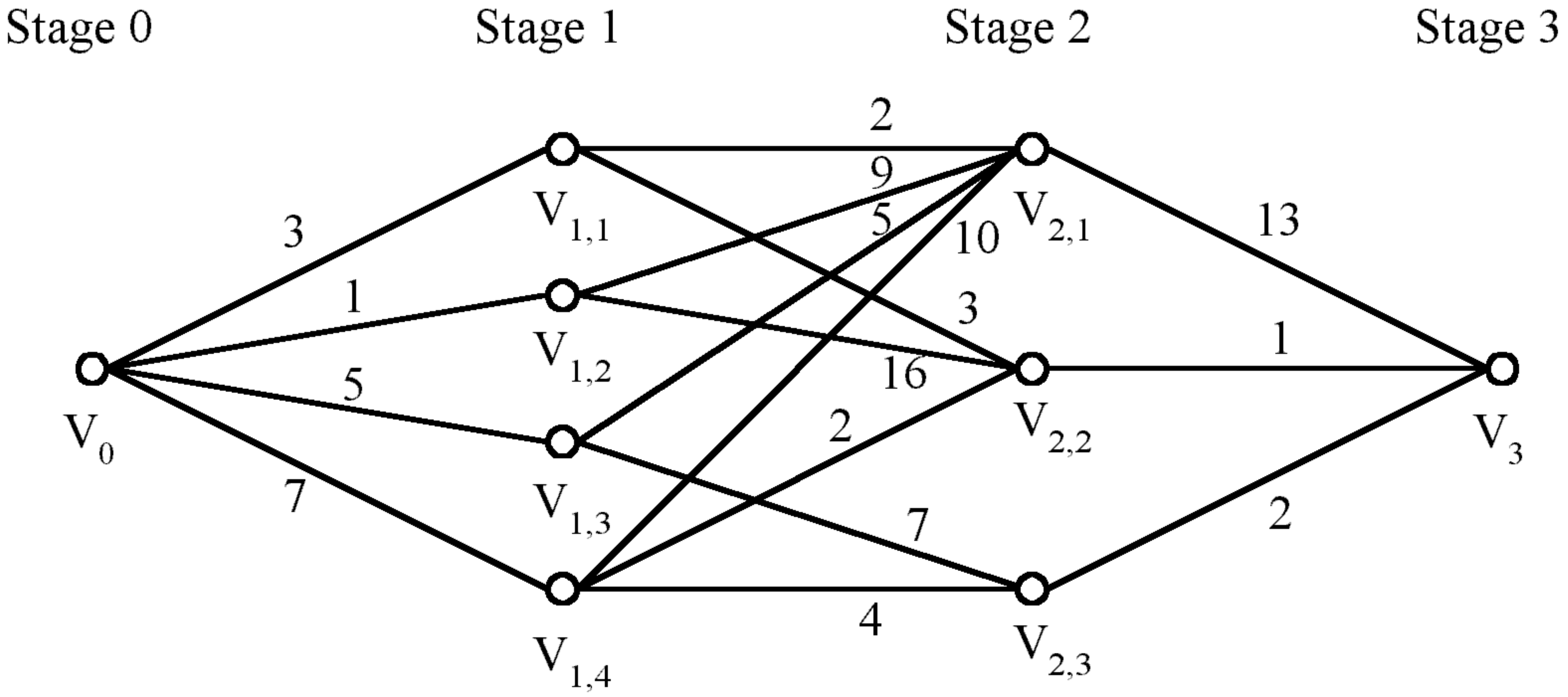


Graph Problems

- What is the shortest path between v0 and v3 in this graph?
- Can you find it with greedy algorithms?

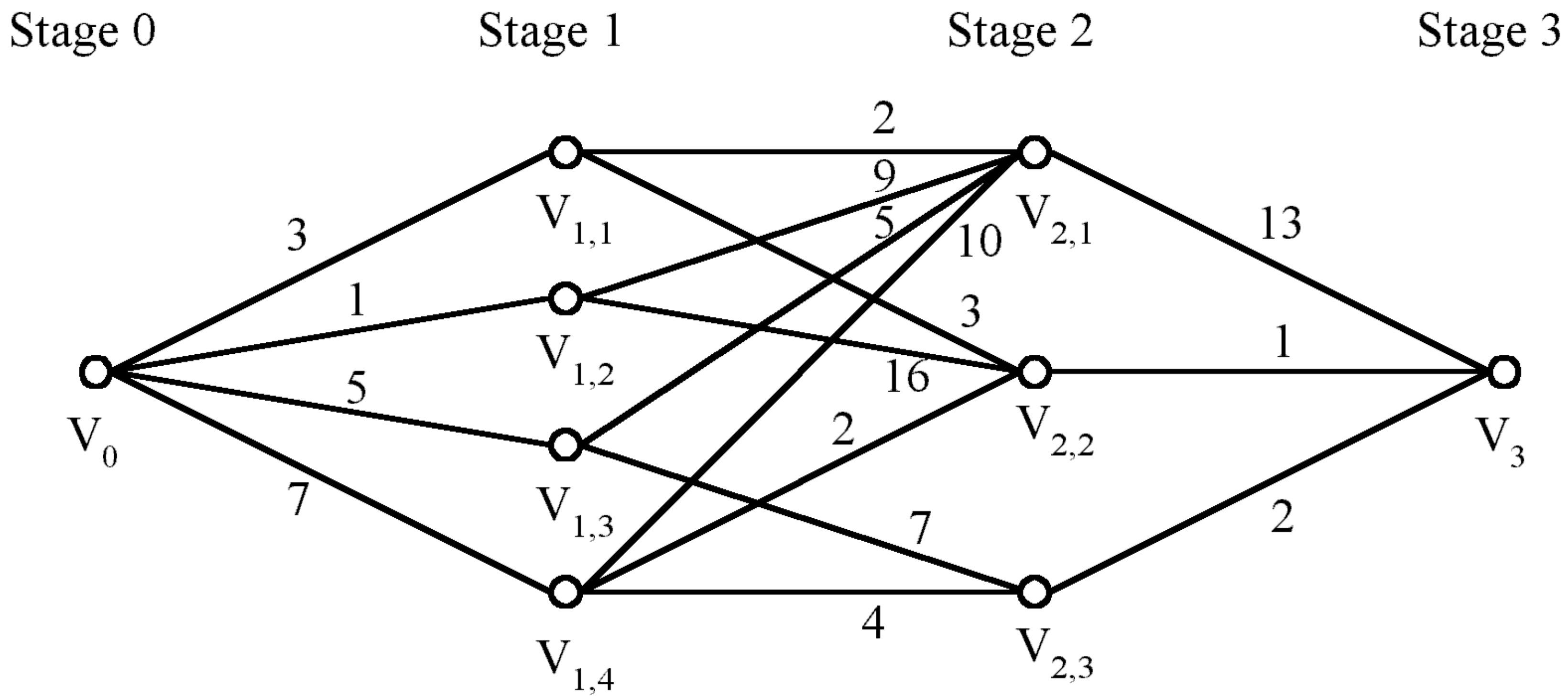


Graph Problems



Graph Problems

- What is the shortest path between v0 and v3 in this graph?
- Can you find it with greedy algorithms?



Change-Making Problem - GENERAL

- Given a value N, if we want to make change for N cents, and we have infinite supply of each of S = { S1, S2, .. , Sm} valued coins, how many ways can we make the change? The order of coins doesn't matter.

- For example, for N = 4 and S = {1,2,3}, there are four solutions: {1,1,1,1},{1,1,2},{2,2},{1,3}. So output should be 4. For N = 10 and S = {2, 5, 3, 6}, there are five solutions: {2,2,2,2,2}, {2,2,3,3}, {2,2,6}, {2,3,5} and {5,5}. So the output should be 5.

Change-Making Problem - GENERAL

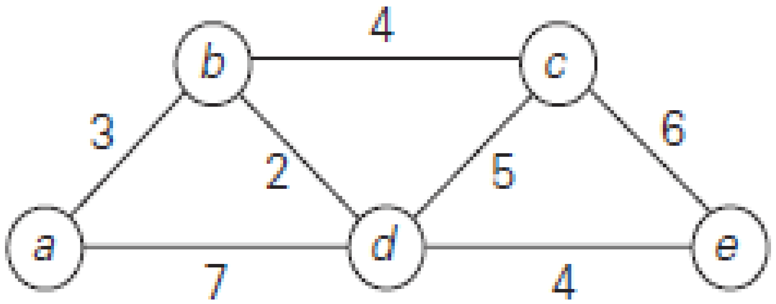
- To count total number solutions, we can divide all set solutions in two sets.
  1) Solutions that do not contain mth coin (or Sm).
  2) Solutions that contain at least one Sm.
  Let count(S[], m, n) be the function to count the number of solutions, then it can be written as sum of count(S[], m-1, n) and count(S[], m, n-Sm).
- Therefore, the problem has optimal substructure property as the problem can be solved using solutions to subproblems.

Dijkstra's Algorithm

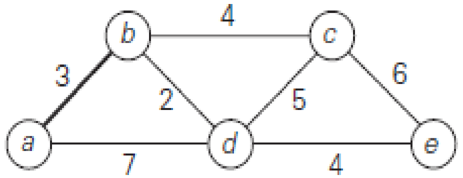
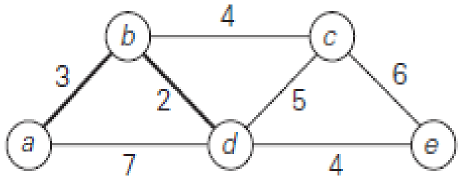
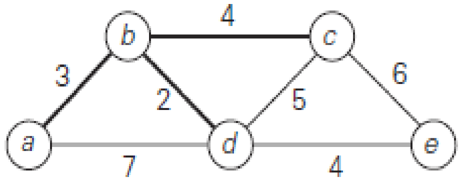
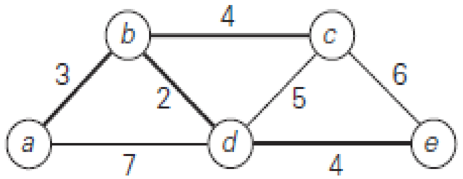
ALGORITHM *Dijkstra*(G, s)

//Dijkstra's algorithm for single-source shortest paths
//Input: A weighted connected graph $G = \langle V, E \rangle$ with nonnegative weights
// and its vertex s
//Output: The length d_v of a shortest path from s to v
// and its penultimate vertex p_v for every vertex v in V
$Initialize(Q)$ //initialize priority queue to empty
for every vertex v in V
  $d_v \leftarrow \infty$; $p_v \leftarrow$ **null**
  $Insert(Q, v, d_v)$ //initialize vertex priority in the priority queue
$d_s \leftarrow 0$; $Decrease(Q, s, d_s)$ //update priority of s with d_s
$V_T \leftarrow \varnothing$
for $i \leftarrow 0$ **to** $|V| - 1$ **do**
  $u^* \leftarrow DeleteMin(Q)$ //delete the minimum priority element
  $V_T \leftarrow V_T \cup \{u^*\}$
  for every vertex u in $V - V_T$ that is adjacent to u^* **do**
    if $d_{u^*} + w(u^*, u) < d_u$
      $d_u \leftarrow d_{u^*} + w(u^*, u)$; $p_u \leftarrow u^*$
      $Decrease(Q, u, d_u)$

Dijkstra’s Algorithm



Dijkstra’s Algorithm

Tree vertices	Remaining vertices	Illustration
a(−, 0)	**b(a, 3)** c(−, ∞) d(a, 7) e(−, ∞)	
b(a, 3)	c(b, 3 + 4) **d(b, 3 + 2)** e(−, ∞)	
d(b, 5)	**c(b, 7)** e(d, 5 + 4)	
c(b, 7)	**e(d, 9)**	
e(d, 9)		

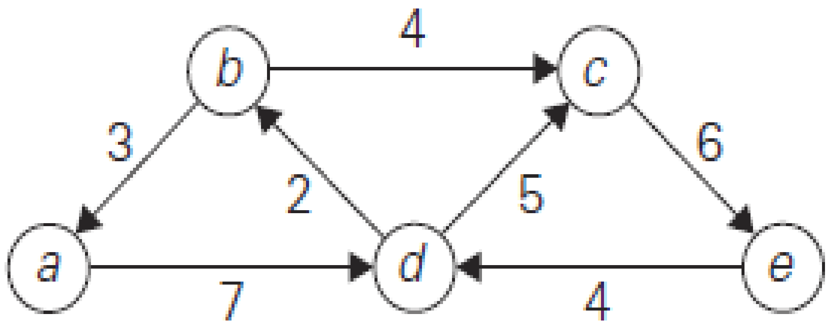
Dijkstra’s Algorithm

The shortest paths (identified by following nonnumeric labels backward from a destination vertex in the left column to the source) and their lengths (given by numeric labels of the tree vertices) are as follows:

from a to b :	$a - b$	of length 3
from a to d :	$a - b - d$	of length 5
from a to c :	$a - b - c$	of length 7
from a to e :	$a - b - d - e$	of length 9

Exercises

- Find the shortest path in the figure starting from a and ending with another letter.



Exercises

- Find the shortest path in the figure starting from a and ending with another letter.