# Chapter 15

# Dynamic Programming

## 15.1 Introduction

## 15.2 Why greedy algorithm may fail to give solution?

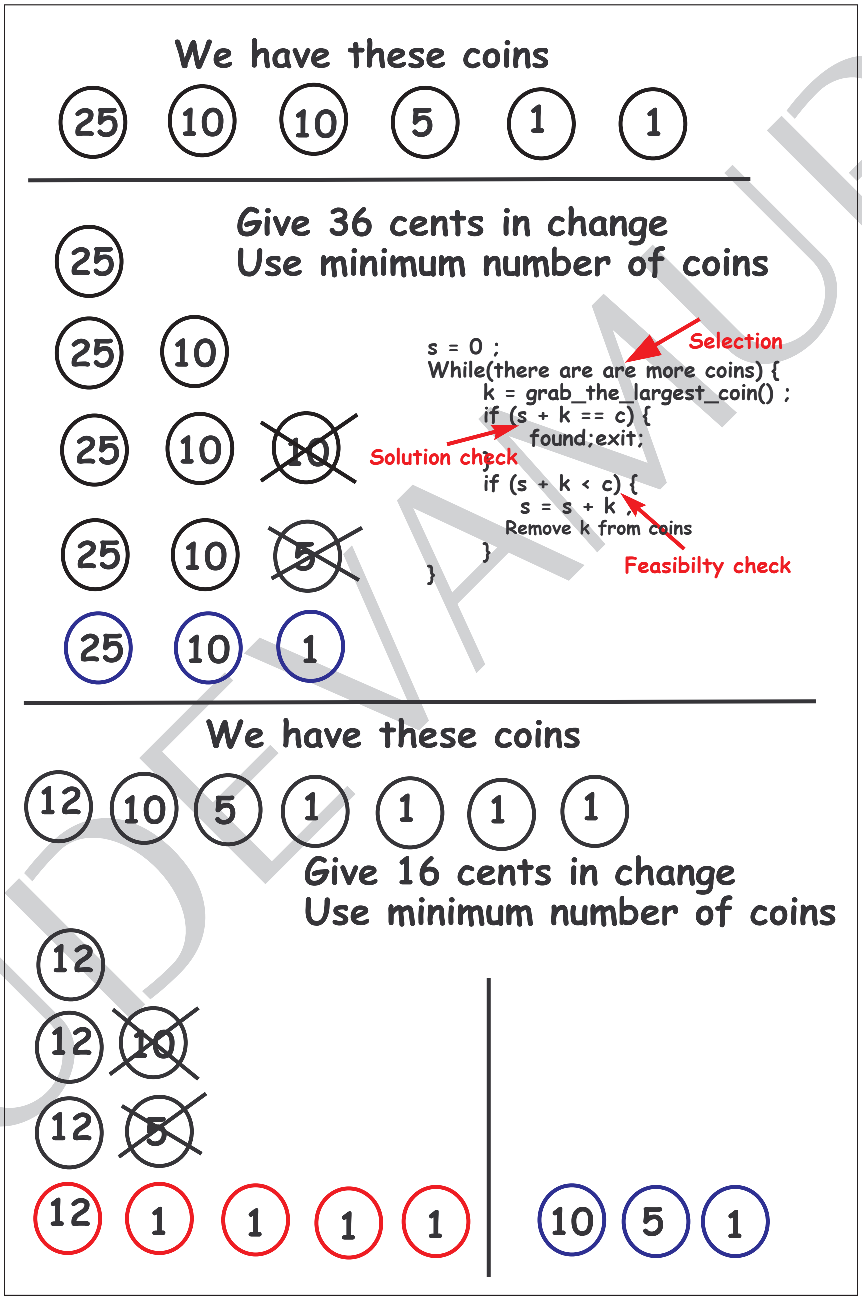



Figure 15.1: Correct and incorrect solutions using greedy algorithm

## 15.3 Fibonacci numbers

### 15.3.1 Fibonacci numbers using divide and conquer

### 15.3.2 Fibonacci numbers using dynamic programming

## 15.4 Coin change problem



Amazon Interview Programming Problem

The `StampDispenser` class represents a postage stamp vending machine. The machine contains stamps of different values. The machine will always have a stamp with a value of 1 cent and the machine will never run out of any type of stamp. The machine should allow a consumer of the class to calculate the minimum number of stamps that the machine can dispense to fill a given request.

*Your task is to complete one of the provided implementations of the* `StampDispenser` *class*: *C++, C#,* or Java.

```
/**
 * Facilitates dispensing stamps for a postage stamp machine.
 */
public class StampDispenser
{
    /**
     * Constructs a new StampDispenser that will be able to dispense the given
     * types of stamps.
     *
     * @param stampDenominations The values of the types of stamps that the
     *     machine should have.  Should be sorted in descending order and
     *     contain at least a 1.
     */
    public StampDispenser(int[] stampDenominations)
    {
    }

    /**
     * Returns the minimum number of stamps that the machine can dispense to
     * fill the given request.
     *
     * @param request The total value of the stamps to be dispensed.
     */
    public int calcMinNumStampsToFillRequest(int request)
    {
        return 0;
    }
}
```

As an example, suppose an instance of `StampDispenser` was created with `stampDenominations`, {90, 30, 24, 10, 6, 2, 1}, and `calcMinNumStampsToFillRequest(int)` was called with request, 34. The call should return 2, as 34 cents can best be filled by one 24 cent stamp and one 10 cent stamp.

Things to keep in mind:

1. Assume that a junior programmer is going to read your code. You should include comments and any other aides that you use to communicate your code to other developers.
2. Optimize the code for speed.
3. The code should compile and work.
4. The code should work for countries with high denomination values where stamp values of 1000 or 9000 are common.



Figure 15.2: Amazon Interview Question

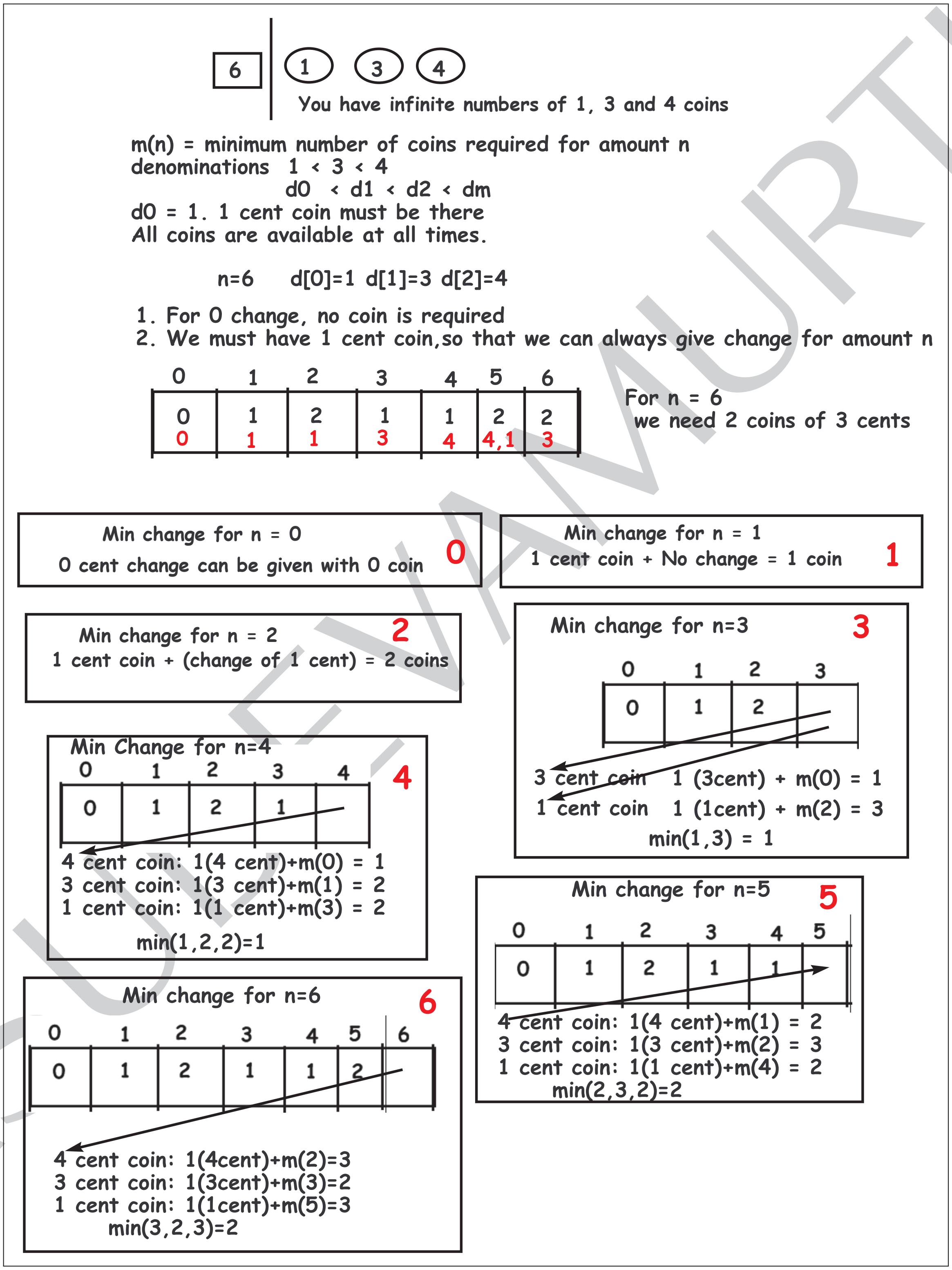



Figure 15.3: Building optimal table using dynamic programming

```java
import java.util.ArrayList;

/**
 * File Name: CoinChange.java.java
 * Dynamic programming
 *
 * @author Jagadeesh Vasudevamurthy
 * @year 2014
 */

/*
 * NOTHING CAN BE CHANGED IN THIS FILE
 */
abstract class CoinChangeBase{
  //I don't know how to write it
  //Override by the concrete class
  abstract protected void computeChange(int n, int [] arrayOfCoinsAvailable) ;

   protected void testBench() {
    {
      System.out.println("------------test1--------------------");
      int[] w = {1,3,4};
      computeChange(6,w) ;
    }
    {
      System.out.println("------------test2--------------------");
      int[] w = {1,2,6,10,24,30,90};
      computeChange(100,w) ;
    }
    {
      System.out.println("------------Amazon stamp-------------------");
      int[] w = {1,2,6,10,24,30,90};
      computeChange(34,w) ;
    }
    {
      System.out.println("------------US coins-------------------");
      int[] w = {1,5,10,25};
      computeChange(24,w) ;
    }
    {
      System.out.println("------------Weird coins-------------------");
      int[] w = {1,5,10,12};
      computeChange(16,w) ;
    }
  }

  public static void main(String[] args) {
```

```
    System.out.println("CoinChangeBase.java STARTS");
    System.out.println("You cannot instantiate CoinChangeBaseclass: CoinChangeBase p = new CoinChangeBase() ; ");
    //CoinChangeBase p = new CoinChangeBase() ;
    System.out.println("CoinChangeBase.java ENDS");
  }
}
```

Give minimum change for 16 cents using coins {1,5,10,12}

| i = | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| m array | 0 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 1 | 2 | 3 | 2 | 3 |
| k array | 0 | 1 | 1 | 1 | 1 | 5 | 1 | 1 | 1 | 1 | 10 | 1 | 12 | 1 | 1 | 5 | 1 |

minimum change for 0 cents can be achieved using 0 coins
minimum change for 1 cents can be achieved using 1 coins
1::Pick coin 1. Current val= 1. Remaining val= 0
minimum change for 2 cents can be achieved using 2 coins
1::Pick coin 1. Current val= 1. Remaining val= 1
2::Pick coin 1. Current val= 2. Remaining val= 0
minimum change for 3 cents can be achieved using 3 coins
1::Pick coin 1. Current val= 1. Remaining val= 2
2::Pick coin 1. Current val= 2. Remaining val= 1
3::Pick coin 1. Current val= 3. Remaining val= 0
minimum change for 4 cents can be achieved using 4 coins
1::Pick coin 1. Current val= 1. Remaining val= 3
2::Pick coin 1. Current val= 2. Remaining val= 2
3::Pick coin 1. Current val= 3. Remaining val= 1
4::Pick coin 1. Current val= 4. Remaining val= 0
minimum change for 5 cents can be achieved using 1 coins
1::Pick coin 5. Current val= 5. Remaining val= 0
minimum change for 6 cents can be achieved using 2 coins
1::Pick coin 1. Current val= 1. Remaining val= 5
2::Pick coin 5. Current val= 6. Remaining val= 0
minimum change for 7 cents can be achieved using 3 coins
1::Pick coin 1. Current val= 1. Remaining val= 6
2::Pick coin 1. Current val= 2. Remaining val= 5
3::Pick coin 5. Current val= 7. Remaining val= 0
minimum change for 8 cents can be achieved using 4 coins
1::Pick coin 1. Current val= 1. Remaining val= 7
2::Pick coin 1. Current val= 2. Remaining val= 6
3::Pick coin 1. Current val= 3. Remaining val= 5
4::Pick coin 5. Current val= 8. Remaining val= 0
minimum change for 9 cents can be achieved using 5 coins

```java
/**
 * File Name: CoinChange.java
 * Dynamic programming
 *
 * @author Jagadeesh Vasudevamurthy
 * @year 2018
 */
class CoinChange extends CoinChangeBase{
  //input to the program
  private int n ; //amount for which change has to be given
  private int[] d ; //array of denominations {1,3,4}
  //Data used for solution
  //YOU CAN HAVE ANY NUMBER OF PRIVATE VARIABLES AND PRIVATE FUNCTIONS
  private static final IntUtil u = new IntUtil();

  CoinChange() {
    //Nothing can be added here
    testBench();
  }

  @Override
  protected void computeChange(int amount, int [] arrayOfCoinsAvailable) {
    //NOTHING CAN BE CHANGED HERE
    n = amount ;
    d = arrayOfCoinsAvailable ;
    solve() ;
  }

  private void solve() {
    //WRITE CODE HERE
    //YOU CAN HAVE ANY NUMBER OF PRIVATE FUNCTIONS

    for (int i = 0; i <= n; ++i) {
      //Change the line below.
      //minimum change for 5 cents can be achieved using 1 coins
      //System.out.println("minimum change for " + i + " cents can be achieved using " + m[i] + " coins") ;
      printSolution(i) ;
    }
  }

  private void printSolution(int p){

  }

  public static void main(String[] args) {
    //NOTHING CAN BE CHANGED BELOW
```

Email
1.Coinchange.pdf
2.Output of the program as text file

```java
    System.out.println("CoinChange problem STARTS");
    CoinChange m = new CoinChange() ;
    System.out.println("CoinChange problem ENDS");
  }
}
```

## 15.5 0/1 Knapsack problem

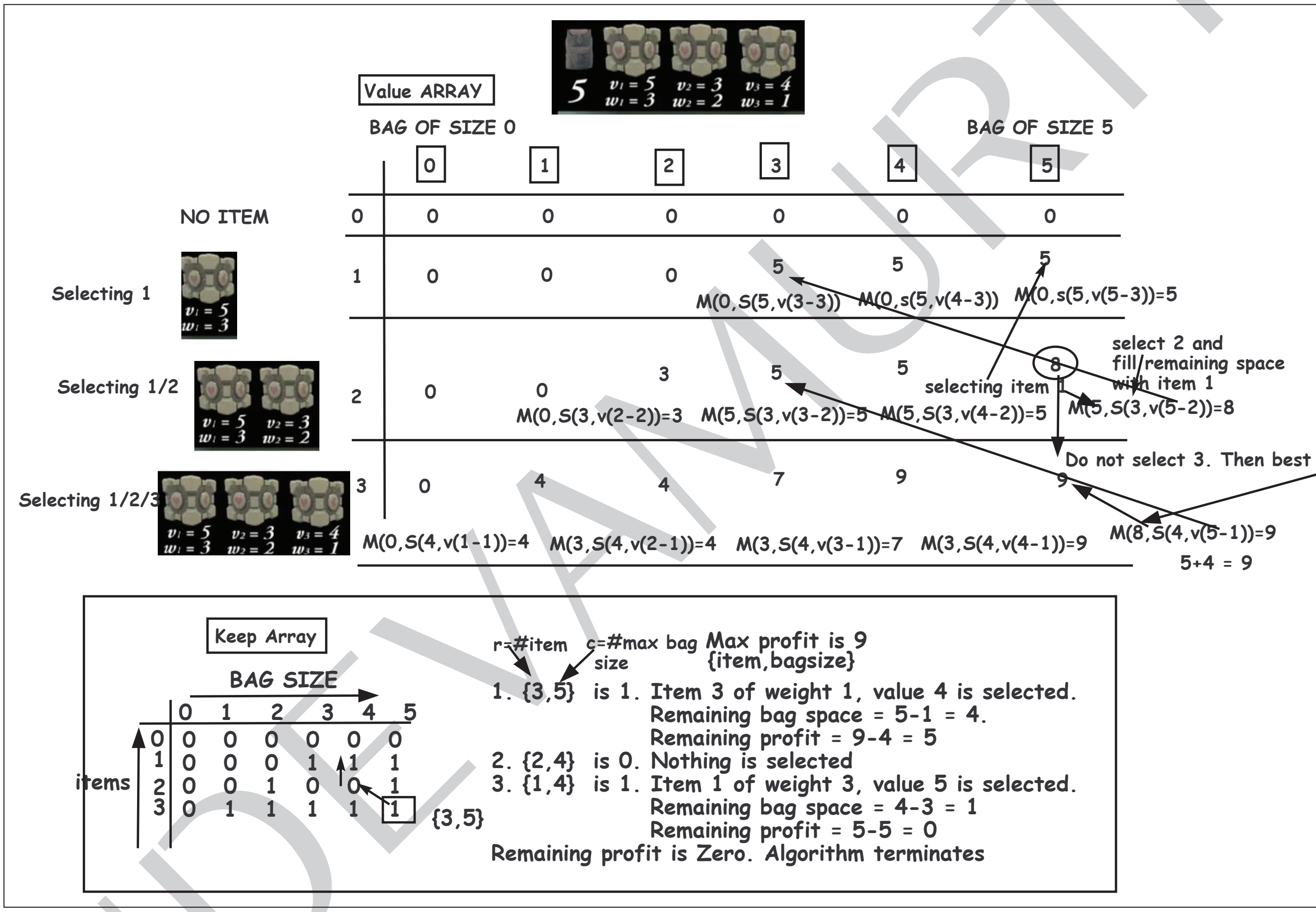



Figure 15.4: Building optimal table using dynamic programming

```
------------ V matrix ----------------
   0   0   0   0   0   0
   0   0   0   5   5   5
   0   0   3   5   5   8
   0   4   4   7   9   9
------------ k matrix ----------------
   0   0   0   0   0   0
   0   0   0   1   1   1
   0   0   1   0   0   1
   0   1   1   1   1   1
i =    1   2   3
w =    3   2   1
v =    5   3   4
Max Value of 9 can obtained from items {3,1} that has values {4+5=9}
```

int[] w = {3,2,1};
int[] v = {5,3,4};
Bag size 5

```
------------ V matrix ----------------
   0   0   0   0   0   0
   0   0  12  12  12  12
   0  10  12  22  22  22
   0  10  12  22  30  32
   0  10  15  25  30  37
------------ k matrix ----------------
   0   0   0   0   0   0
   0   0   1   1   1   1
   0   1   0   1   1   1
   0   0   0   0   1   1
   0   0   1   1   0   1
i =    1   2   3   4
w =    2   1   3   2
v =   12  10  20  15
Max Value of 37 can obtained from items {4,2,1} that has values {15+10+12=37}
```

int[] w = {2,1,3,2};
int[] v = {12,10,20,15};
Bag size 5

```
------------ V matrix ----------------
   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0  10  10  10  10  10  10
   0   0   0   0  40  40  40  40  40  50  50
   0   0   0   0  40  40  40  40  40  50  70
   0   0   0  50  50  50  50  90  90  90  90
------------ k matrix ----------------
   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   1   1   1   1   1   1
   0   0   0   0   1   1   1   1   1   1   1
   0   0   0   0   0   0   0   0   0   0   1
   0   0   0   1   1   1   1   1   1   1   1
i =    1   2   3   4
w =    5   4   6   3
v =   10  40  30  50
Max Value of 90 can obtained from items {4,2} that has values {50+40=90}
```

int[] w = {5,4,6,3};
int[] v = {10,40,30,50};
Bag size 10

Figure 15.5: Animation of 0/1 knapsack problem using dynamic programming

## 15.6 Shortest path algorithm

### 15.6.1 Gready algorithm that does not work

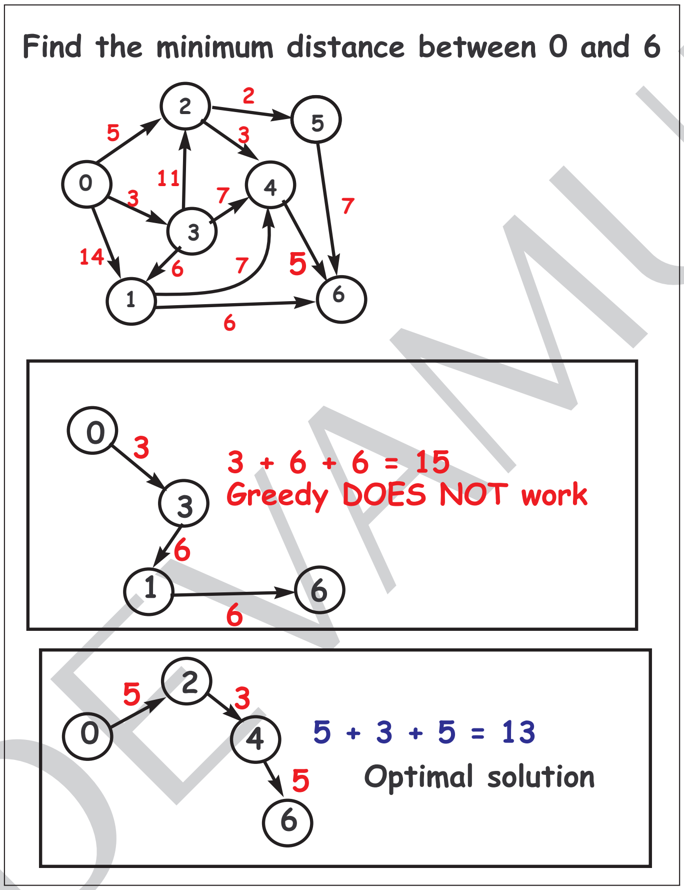



Figure 15.6: Greedy algorithm that fails to work

### 15.6.2 Topological sorting

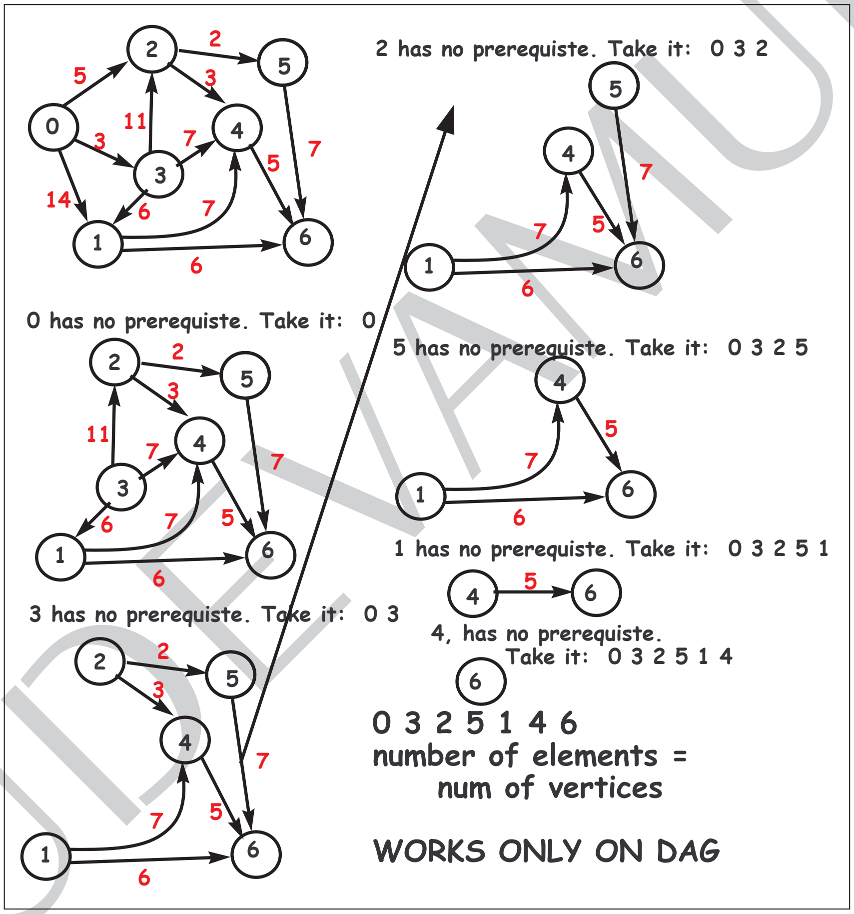



Figure 15.7: Finding all prerequisites/predecessors using toplogical sort

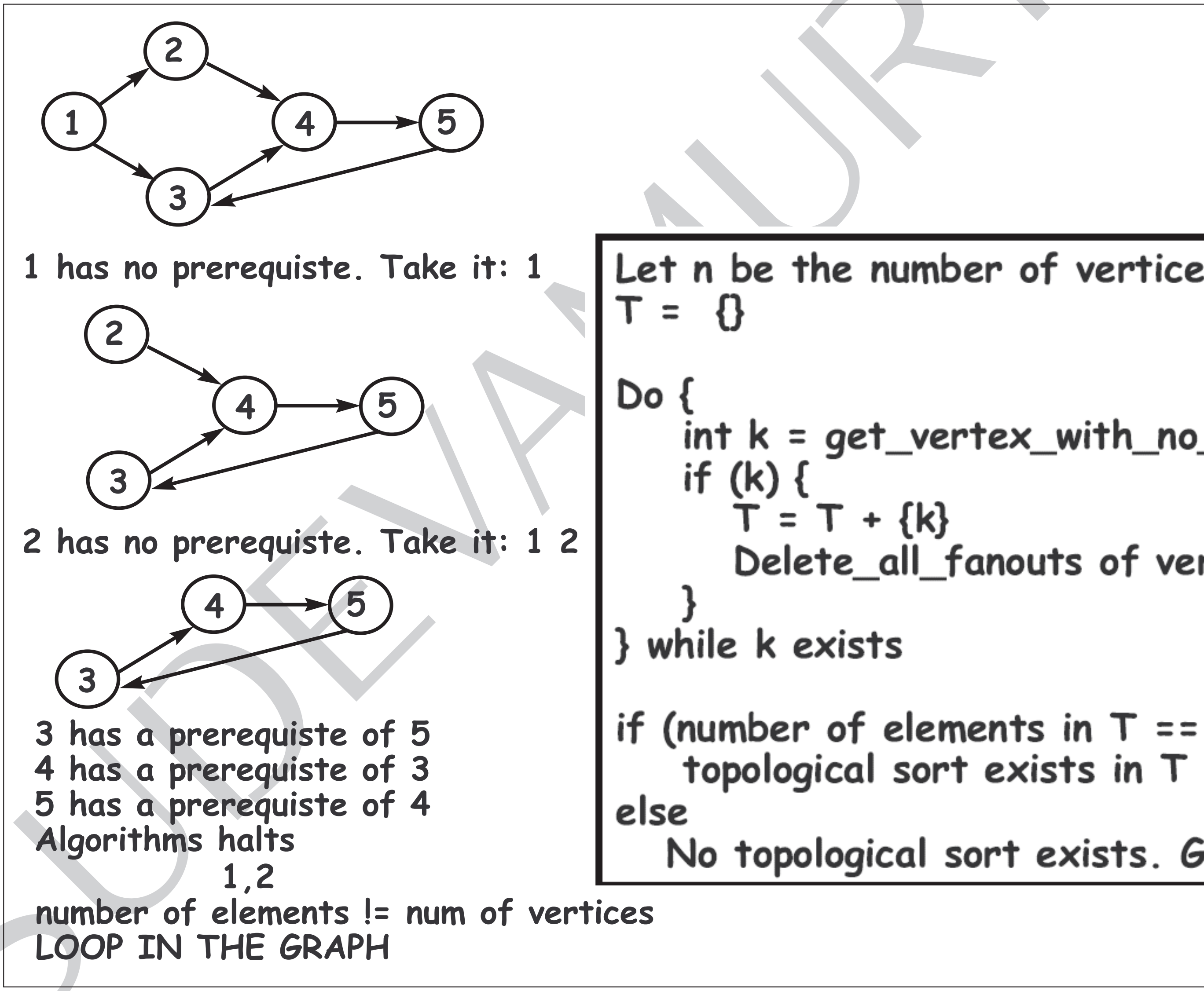



Figure 15.8: Toplogical sort algorithm

### 15.6.3 Shortest path algorithm using dynamic programming

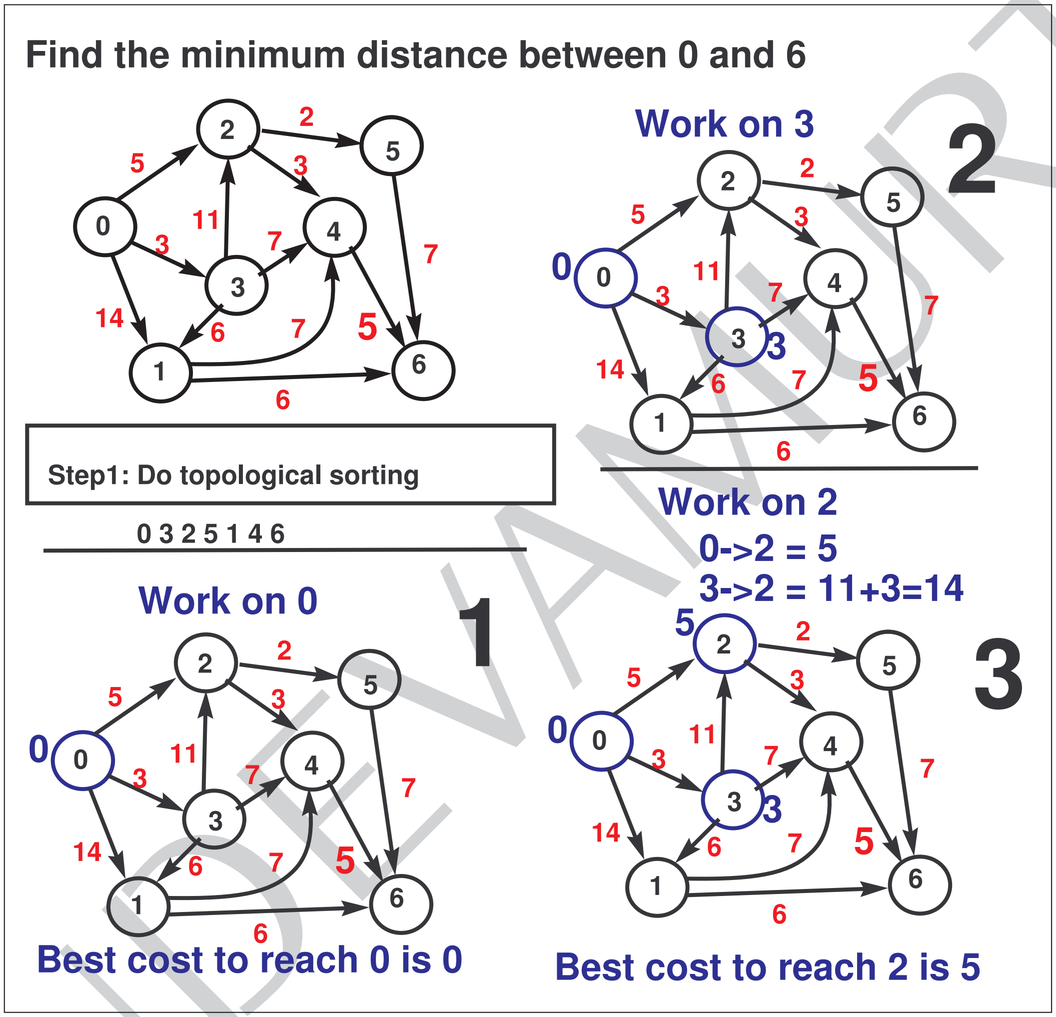



Figure 15.9: Shortest path algorithm using dynamic programming

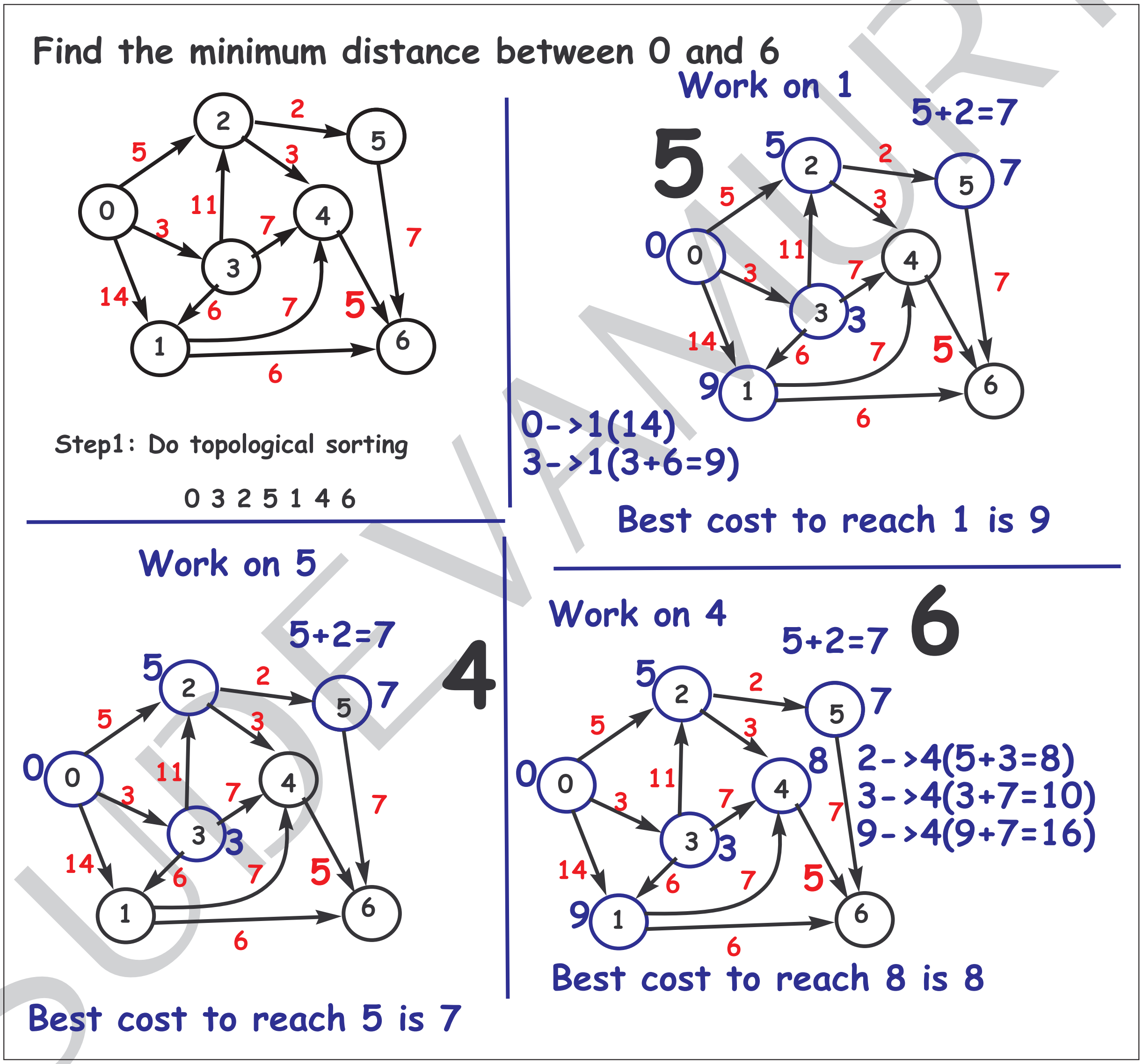



Figure 15.10: Shortest path algorithm using dynamic programming(contd)

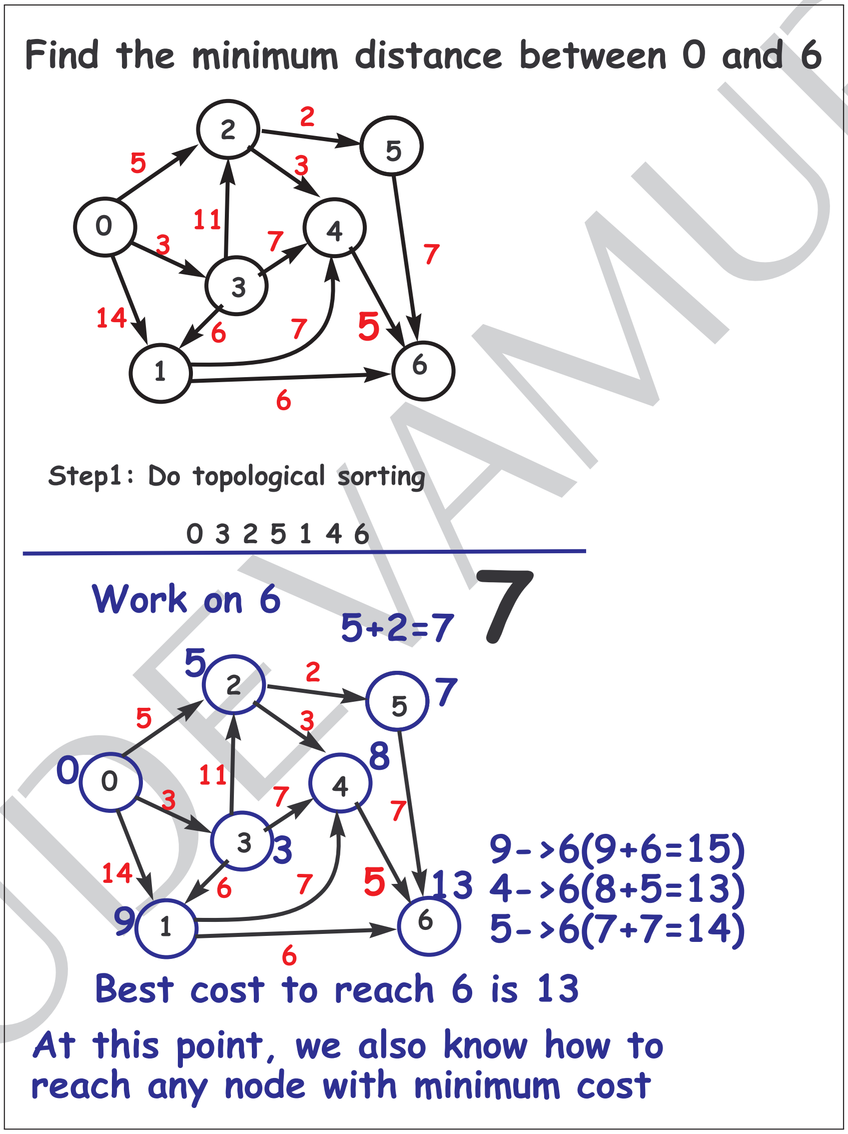
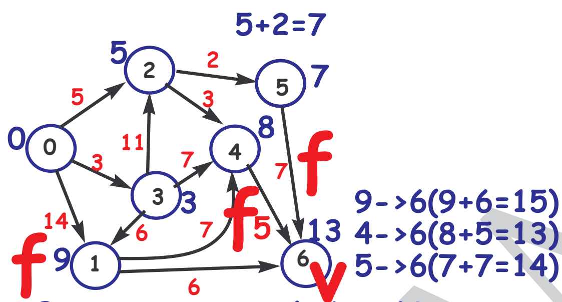



Figure 15.11: Shortest path algorithm using dynamic programming(contd)




Best cost to reach 6 is 13

```
Do topological sort on the graph
Let the answer be in array t of size n

create a weight array w of size n
for(i = 0; i < n; i++) w[i] = 0 ;

for(i = 1; i < n; i++) {
     v = t[i] ;
     int min = INF ;
     for_each_fanins(v,f) {
          if (w[f] + weight(f,v) < min)
               min = w[f]+weight(f,v);
     }end_for_each_fanins
     w[v] = min ;
}
```

Theorem: Must be Available
and must be minimum
Basis of dynamic programming

Given by user

Figure 15.12: Algorithm for shortest path algorithm using dynamic programming

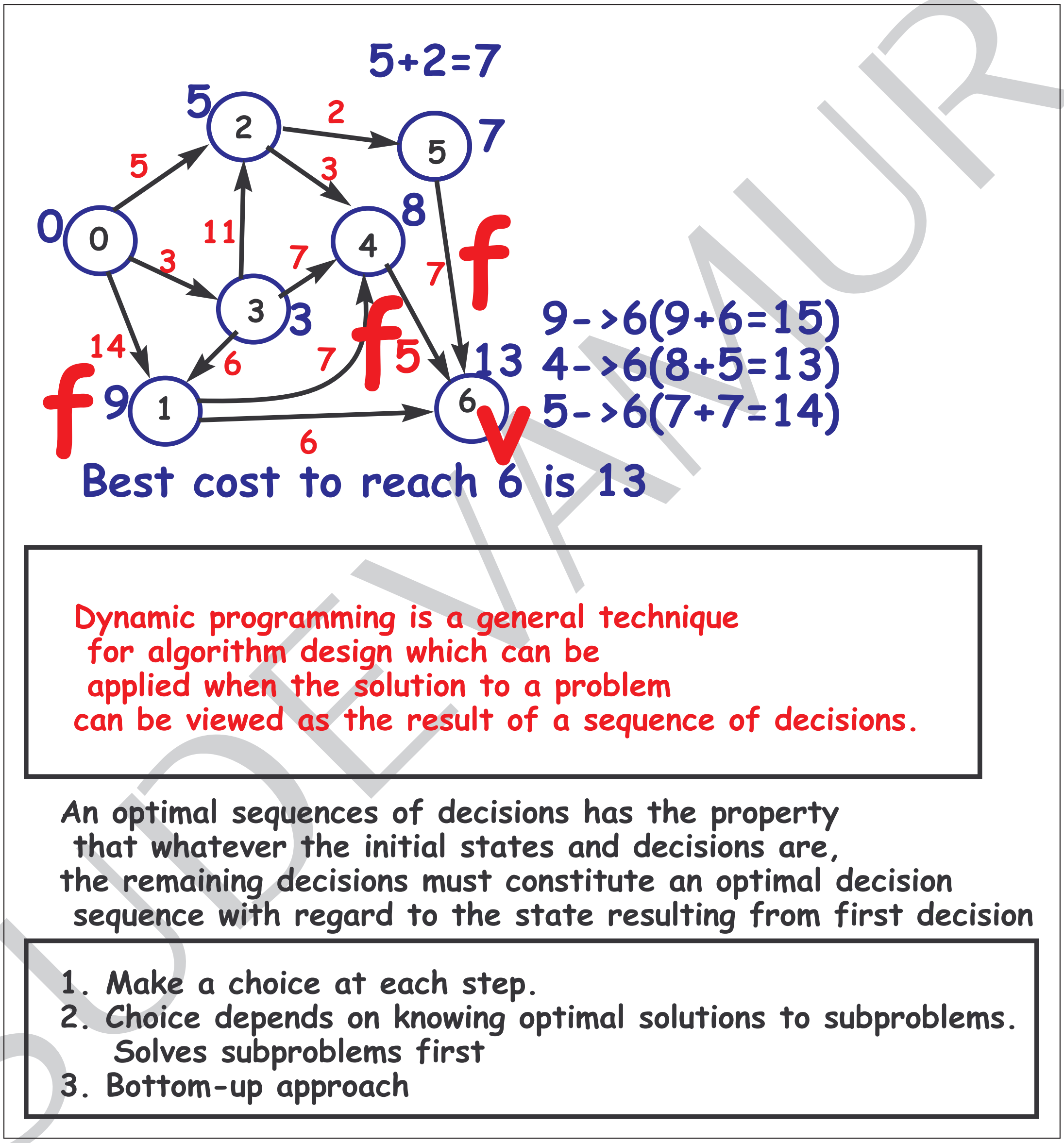



Figure 15.13: Principle of dynamic programming

## 15.7 Problem set

**Problem 15.7.1.**



Solve the problem shown in figure 15.14

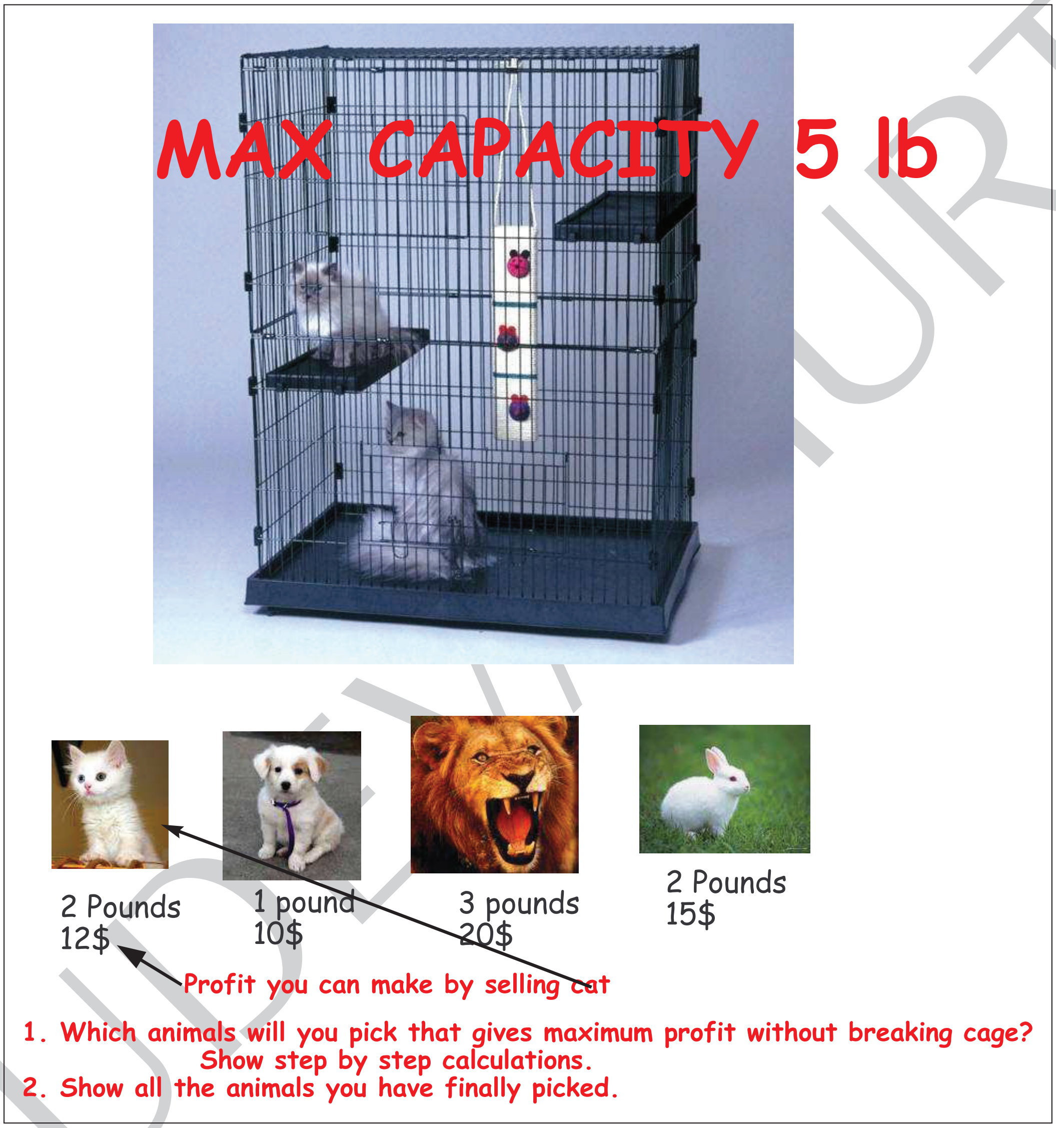



Figure 15.14: Which animals will you pick without breaking the cage?

