Werkcollege 9
Dynamic Programming

Jan van Rijn

Leiden Institute of Advanced Computer Science

7 april 2011

Opgave 1

Levitin 8.1.2

Bereken $\binom{6}{3}$ door het algoritme uit het boek toe te passen.

```
1   ALGORITHM Binomial( n, k ){
2     // computes n choose k by the dynamic programming
          algorithm
3     // Input: A pair of non-negative integers n ≥ k ≥ 0
4     // Output: the value of n choose k.
5     for( i = 0 to n ) do
6       for( j = 0 to min(i, k) ) do
7         if( j == 0 || j == i )
8           C[i][j] = 1;
9         else
10          C[i][j] = C[i-1][j-1] + C[i-1][j];
11    return C[n][k];
12  }
```

Opgave 1

```
1   ALGORITHM Binomial( n, k ){
2     // same as previous algorithm
3     for( i = 0 to k ) do
4       for( j = i to n ) do
5         if( j == 0 || j == i )
6           C[j][i] = 1;
7         else if( j > i )
8           C[j][i] = C[j-1][i-1] + C[i-1][j];
9      return C[n][k];
10  }
```

Opgave 2

Levitin 8.1.10

Twee teams, A en B, spelen een *serie* wedstrijden tot een van de teams n wedstrijden heeft gewonnen. Neem aan dat de kans dat A een wedstrijd wint altijd hetzelfde is, namelijk p $(0 \leq p \leq 1)$. De kans dat A de wedstrijd verliest is q. $q = 1 - p$. (Merk op dat er geen gelijke spelen zijn.) Laat $P(i, j)$ de kans zijn dat A de serie wint wanneer A nog i wedstrijden moet winnen, en B nog j wedstrijden moet winnen.

- Stel een recurrente betrekking op voor $P(i, j)$ die gebruikt kan worden in een Dynamic Programming algoritme.
- Gebruik deze betrekking op de kans te berekenen dat A de serie wint als de kans dat A een wedstrijd wint 0.4 is. $(p = 0.4)$
- Schrijf in pseudo code een dynamic programming algoritme voor dit probleem.

Opgave 2

```
1   ALGORITHM SeriesWinChance( n, p ){
2     q = 1 - p;
3     for( j = 1 to n ){
4       P[0][j] = 1.0;
5     }
6     for( i = 1 to n ){
7       P[i][0] = 0.0;
8       for( j = 1 to n ){
9         P[i][j] = p * P[i-1][j] + q * P[i][j-1];
10      }
11    }
12    return P[n][n];
13  }
```

Opgave 3

Levitin 8.4.1

Bereken de oplossing voor de volgende instantie van het knapzakprobleem.

item	weight	value	
1	3	25	
2	2	20	
3	1	15	$W = 6$
4	4	40	(capacity)
5	5	50	

Opgave 3

```
1   ALGORITHM DPKnapsackProblem( w[1,...,n], v[], W ){
2     for( i = 0 to n ){
3       V[i][0] = 0;
4     }
5     for( j = 1 to n ){
6       V[0][j] = 0;
7     }
8     for( i = 1 to n ){
9       for( j = 1 to n ){
10        if( j - w[i] >= 0 ){
11          V[i][j] = max( V[i-1][j], v[i]+V[i-1][j-w[i]]
                );
12        } else {
13          V[i][j] = V[i-1][j];
14      }
15    }
16    return V[n][W];
17  }
```

Opgave 4

Recursieve functie:

```
1   bool balans( int n, int W ){
2     if(  n== 1 ){
3       if( ( W == 0 ) || ( W == gewicht[1] ) ) return
            true;
4       else return false;
5     } else {
6       bool metDitGewicht = balans( n-1, W-gewicht[n] );
7       bool zonderDitGewicht = balans( n-1, W );
8       if( metDitGewicht == true || zonderDitGewicht ==
            true )
9         return true;
10      else return false;
11    }
12  }
```

Opgave 4

Recurrente betrekking:

wegen[i][j] =	wegen[i-1][j] or	
	wegen[i-1][j-gewicht[i]]	als $i > 1$ en $j >= $ gewicht[i]
wegen[i][j] =	wegen[i-1][j]	als $i > 1$ en $j <$ gewicht[i]
wegen[1][j] =	true als $j = 0$ of $j = g1$	
wegen[i][0] =	true als $i > 1$;	

Opgave 4

```
1   ALGORITHM useWeights( int n, int W ){
2     for( i = 0 to n ){
3       wegen[i][0] = true;
4     }
5     for( j = 1 to W ){
6       if( j == gewicht[1] ) wegen[1][j] = true;
7       else wegen[1][j] = false
8     }
9     for( i = 2 to n ){
10      for( j = 1 to W ){
11        bool metDitGewicht = false;
12        if ( j - gewicht[i] > 0 )
13          bool metDitGewicht = wegen( i-1, W-gewicht[n]
                );
14        bool zonderDitGewicht = wegen( i-1, W );
15        wegen[i][j] = ( metDitGewict ||
              zonderDitGewicht );
16      }
17    }
18    return wegen[n][W];
19  }
```

Opgave 5

```
1   double winst( int i, int j ){
2     if( i == 0 ) return 0; // onderaan het rooster,
          geen winst.
3     else if( rooster[i][j] == '.' )
4       // laat balletje zakken
5       return ( winst[i+1][j] );
6     else if( rooster == '*' )
7       // balletje valt links of rechts
8       return 0.5*winst( i-1, j-1 ) + 0.5*winst( i-1, j
            +1 );
9     else
10      // we hebben geld gewonnen!
11      return rooster[i][j];
12  }
13  int win = 0;
14  for( int j = 0; i < n; j++ ){
15    double dezeRij = winst( m, j );
16    if( dezeKolom > win ) win = dezeRij;
17  }
```

Opgave 5

```
1   ALGORITHM maxWinst( int n, int m ){
2     for( j = 1 to n )
3       D[0][j] = 0;
4     for( i = 1 to m ){
5       for( j = 1 to n ){
6         if( rooster[i][j] == '.' )
7           D[i][j] = D[i-1][j];
8         else if( rooster[i][j] == '*' )
9           D[i][j] = 0.5 * D[i-1][j-1] + 0.5 * D[i-1][j
                +1];
10        else
11          D[i][j] = rooster[i][j];
12      }
13    }
14  }
```

De waarde die we zoeken staat nu in rij `n` van `D`.