SC2001/CX2101
Algorithm Design and Analysis

Tutorial 4
Dynamic Programming

(Weeks 10)

This tutorial helps you develop skills in the learning outcome of the course: "Able to design algorithms using suitable strategies (dynamic programming, etc) to solve a problem, able to analyse the efficiencies of different algorithms for problems like optimal sequencing for matrix multiplication, the longest common subsequence, etc".

Question 1

Find the length of the longest common subsequence and a longest common subsequence of CAGAG and ACTGG by the dynamic programming algorithm in the lecture notes.

	1	2	3	4	5
x	C	A	G	A	G
y	A	C	T	G	G

c		A	C	T	G	G
	0	0	0	0	0	0
C	0	0	1	1	1	1
A	0	1	1	1	1	1
G	0	1	1	1	2	2
A	0	1	1	1	2	2
G	0	1	1	1	2	3

<table>
<tr><th>h</th><th></th><th>A</th><th>C</th><th>T</th><th>G</th><th>G</th></tr>
<tr><td></td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr>
<tr><td>C</td><td>|</td><td>|</td><td>\</td><td>—</td><td>—</td><td>—</td></tr>
<tr><td>A</td><td>|</td><td>\</td><td>|</td><td>|</td><td>|</td><td>|</td></tr>
<tr><td>G</td><td>|</td><td>|</td><td>|</td><td>|</td><td>\</td><td>\</td></tr>
<tr><td>A</td><td>|</td><td>\</td><td>|</td><td>|</td><td>|</td><td>|</td></tr>
<tr><td>G</td><td>|</td><td>|</td><td>|</td><td>|</td><td>\</td><td>\</td></tr>
</table>

```
for i = 1 to n
   for j = 1 to m
      if x[i] == y[j]  {
         c[i][j] = c[i-1][j-1] + 1;
         h[i][j] = '\';   }
      else if c[i-1][j] >= c[i][j-1] {
         c[i][j] = c[i-1][j];
         h[i][j] = '|';   }
      else {
        c[i][j] = c[i][j-1];
         h[i][j] = '—';   }
```

LCS(5,5) = 3

<table>
<tr><th>h</th><th></th><th>A</th><th>C</th><th>T</th><th>G</th><th>G</th></tr>
<tr><td></td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr>
<tr><td>C</td><td>|</td><td>|</td><td>\</td><td>—</td><td>—</td><td>—</td></tr>
<tr><td>A</td><td>|</td><td>\</td><td>|</td><td>|</td><td>|</td><td>|</td></tr>
<tr><td>G</td><td>|</td><td>|</td><td>|</td><td>|</td><td>\</td><td>\</td></tr>
<tr><td>A</td><td>|</td><td>\</td><td>|</td><td>|</td><td>|</td><td>|</td></tr>
<tr><td>G</td><td>|</td><td>|</td><td>|</td><td>|</td><td>\</td><td>\</td></tr>
</table>

The subsequence:

C	**G**	**G**

Question 2

The H-number H(n) is defined as follows:

H(0) =1, and for n>0:

$H(n) = H(n-1) + H(n-3) + H(n-5)+ \ldots.+H(0)$ when n is odd

$H(n) = H(n-2) + H(n-4) + H(n-6)+ \ldots.+H(0)$ when n is even.

a) Give a recursive algorithm to compute H(n) for an arbitrary n as suggested by the recurrence equation given for H(n). Draw the tree that represents the recursive calls made when H(8) is computed.

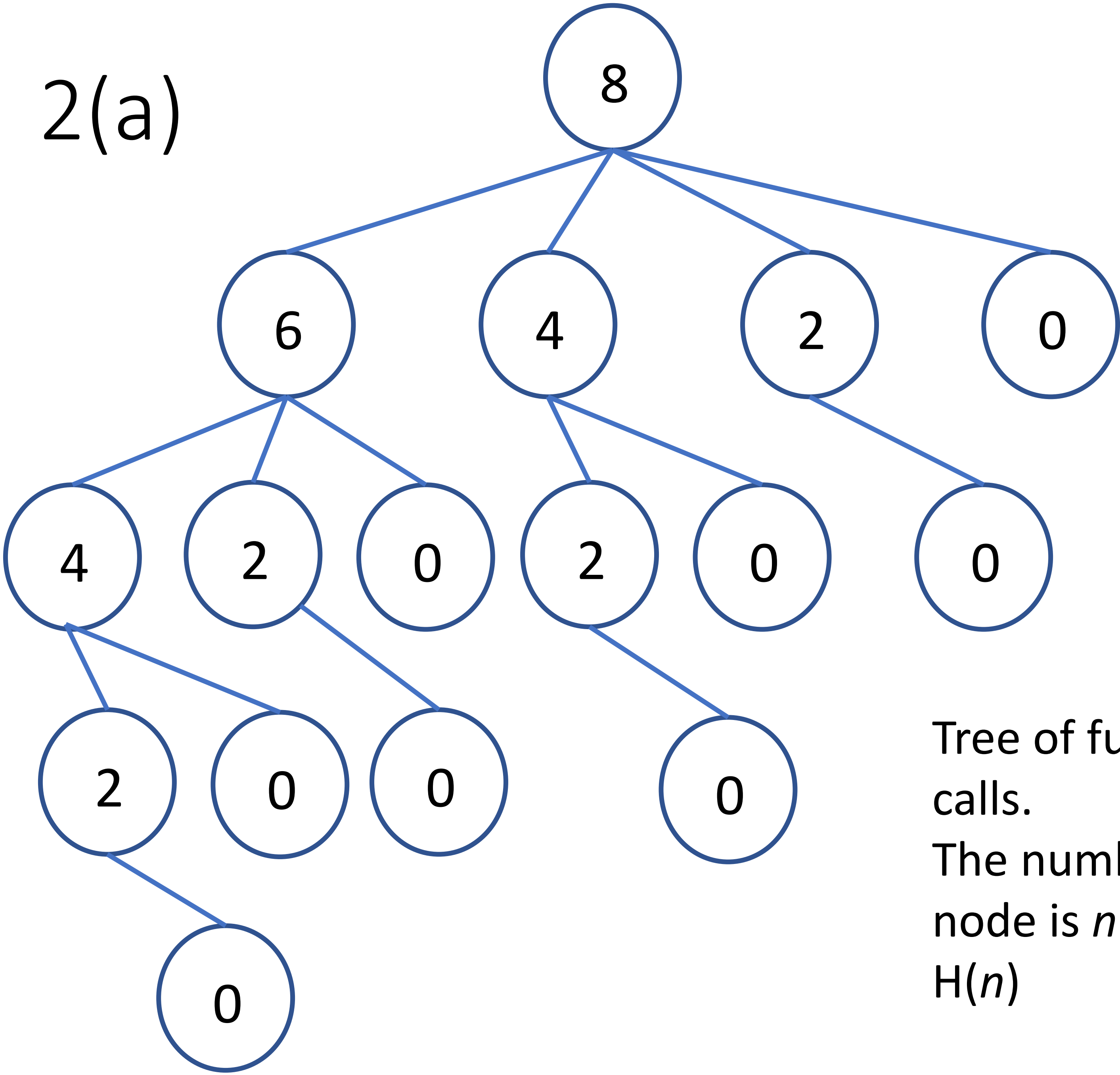
b) Draw the subproblem graph for H(8) and H(9).

c) Write an iterative algorithm using the dynamic programming approach (bottom-up). What are the time and space required?

Question 2(a)

```
int  hn( int n) {
{    if (n == 0)    return 1;
     else {
           S = 0;
           if (n mod 2)     j=n-1;      else  j=n-2;
           for (k = 0; k <= j; k = k+2)
                 S += hn(k);
      }
      return S;
}
```

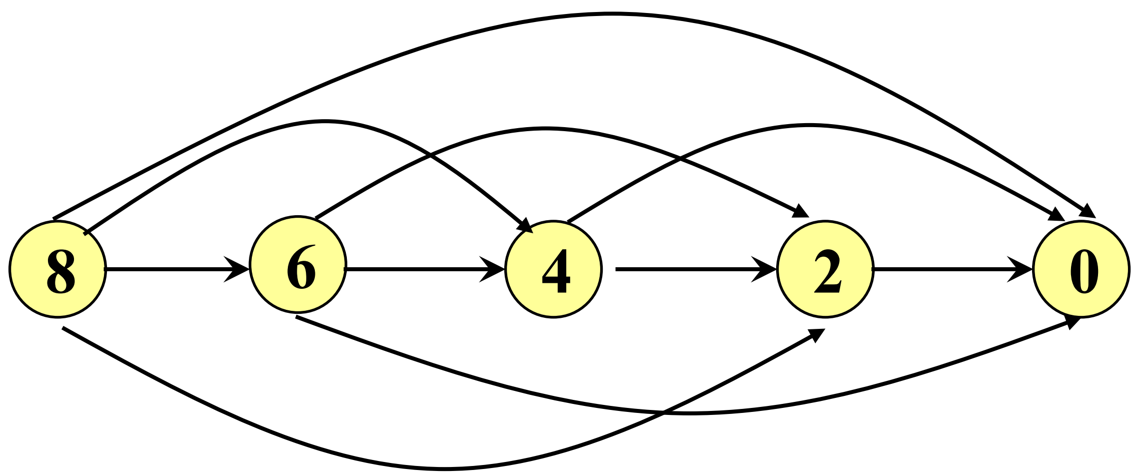
2(a)



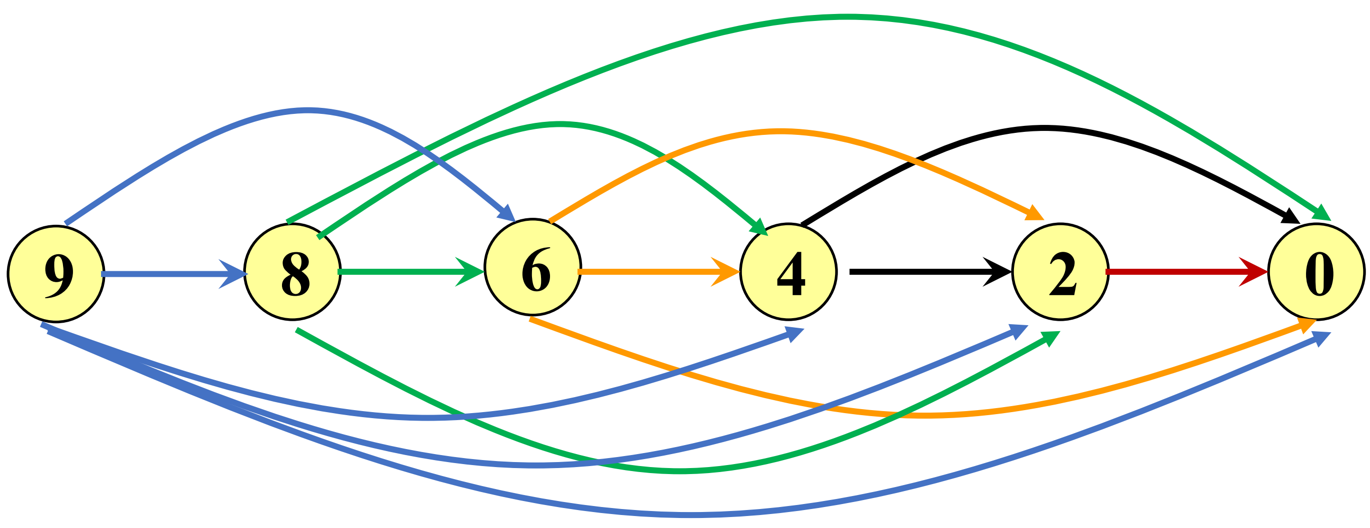
Tree of function calls.
The number in a node is n for a call H(n)

2(b)

The subproblem graph for H(8)



The subproblem graph for H(9)



2(c)

```
int  hn_DP(int n)
{     // Make use of an array S[0..n]
      S[0]=1;
      for (i = 1; i<=n; i++) {
            S[i] = 0;
            if (i mod 2)    j = i-1;    else  j=i-2;
            while (j>=0) { S[i] += S[j]; j-=2;};
       }
       return S[n];
}
```

Space Complexity: $O(n)$.   Time complexity: $O(n^2)$

Question 3

The binomial coefficients can be defined by the recurrence equation:

$C(n, k) = C(n-1, k-1) + C(n-1, k)$	for $n > 0$ and $k > 0$
$C(n, 0) = 1$	for $n >= 0$
$C(0, k) = 0$	for $k > 0$

$C(n, k)$ is also called "*n* choose *k*". This is the number of ways to choose *k* distinct objects from a set of *n* objects.

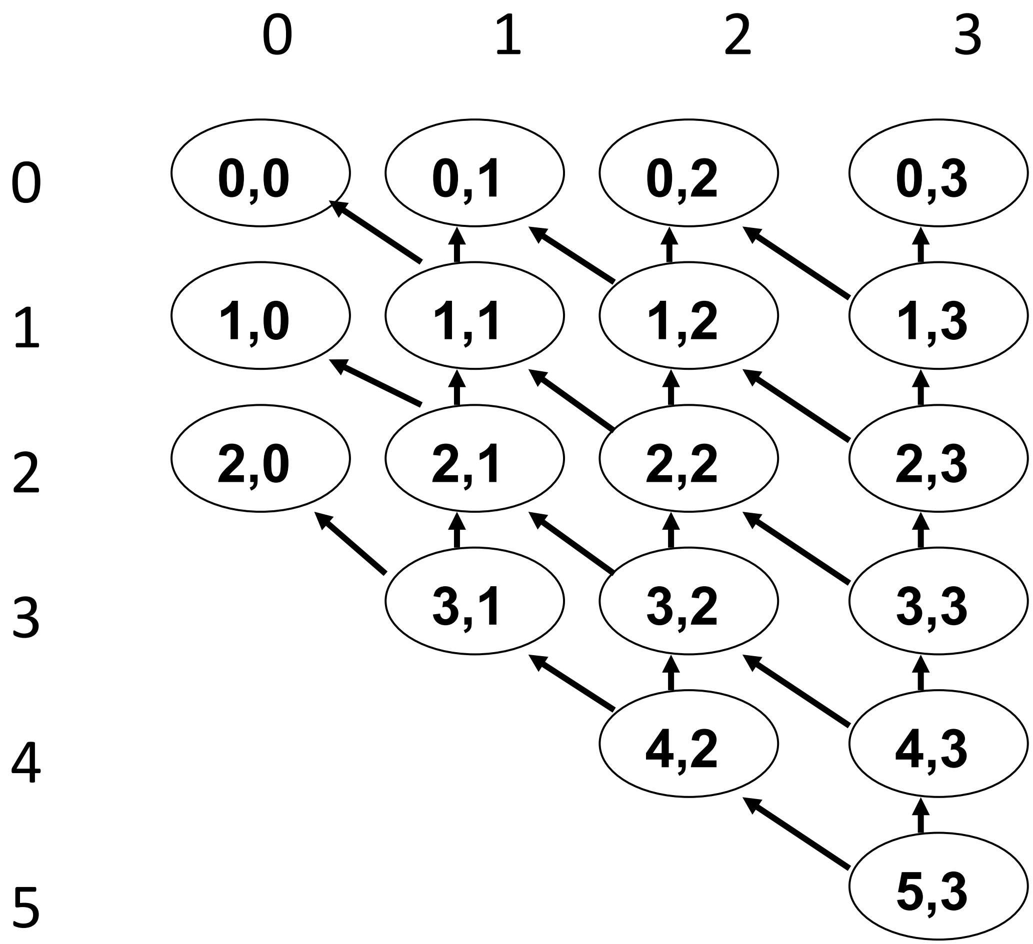
3(a)

Give a recursive algorithm as suggested by the recurrence equation given for C(*n*, *k*).

```
int C(int n, int k)
{
        if (k == 0)   return 1;
        if (n == 0) return 0;

        return C(n – 1, k – 1) + C(n – 1, k);
}
```

3(b) Draw the subproblem graph for C(5, 3).



3(c)

Write a recursive algorithm using the dynamic programming approach (top-down) stating the data structure used for the dictionary.

Use dictionary: int dic[n+1][k+1];
// initialised to –1 in all entries

```
int C(int n, int k, int [] [] dic)
{     int c1, c2;

      if (k == 0)   {
            dic[n][0] = 1;
            return 1;  }
      if (n == 0) {
            dic[0][k] = 0;
            return 0;   }

      if (dic[n – 1][k – 1] == -1)
            c1 = C(n – 1, k – 1);
      else c1 = dic[n – 1][k – 1] ;
      if (dic[n – 1][k] == -1)
            c2 = C(n – 1, k);
      else c2 = dic[n – 1][k] ;

      dic[n][k] = c1 + c2;
      return dic[n][k];
}
```

Time complexity: O(nk)
Space complexity: O(nk)

3(d)

Write an iterative algorithm using the dynamic programming approach (bottom-up).

```
int C(int n, int k, int [] [] dic)
{    int dic[n+1][k+1];

     For (i = 1;  i<= k;  i++)  dic[0][i] = 0;
     For (i = 0;  i<= n;  i++)  dic[i][0] = 1;
     For (i = 1;  i<= n;  i++)
          For (j = 1;  j<= k;  j++)
               dic[i][j] = dic[i-1][j-1] + dic[i-1][j];

    Return dic[n][k];
}
```

Time complexity: O(nk)
Space complexity: O(nk)

```
int C(int n, int k, int [] [] dic)  // more optimized
{     int dic[n+1][k+1];

      For (i = 1;  i<= k;  i++)  dic[0][i] = 0;
      For (i = 0;  i<= n-k;  i++)  dic[i][0] = 1;
      For (i = 1;  i<= n;  i++)
            For (j = max(i-(n-k), 1);  j<= k;  j++)
                  dic[i][j] = dic[i-1][j-1] + dic[i-1][j];

      Return dic[n][k];
}
```