University of Zurich
Department of Informatics
Database Technology Group
Prof. Dr. Michael Böhlen

Informatics II
Exercise 11

March 16, 2022

Dynamic Programming

Task 1

1. Coin change problem can be solved by

   A. Greedy algorithm, can obtain the global optimal solution

   B. Divide and conquer recursion, can obtain the global optimal solution

   C. Dynamic programming, can obtain the global optimal solution

   D. Greedy algorithm, can not obtain the global optimal solution

Solution: D

2. Consider for coin change of amount 14 with coins 10, 7, 2, 1. Give a solution with least coins by dynamic programming.

Solution:
f(14) = min1+f(4), 1+f(7), 1+f(12), 1+f(13)
f(4) = min1+f(2), 1+f(3)
f(2) = 1
f(3) >= 1
then f(4) = 2
f(7) = 1
f(12) >= 1
f(13) >= 1
then f(14) = 2
So, we use 2 coins valued 7 for amount 14

Task 2

Consider a 2-D matrix M. Value of each cell of M is a wall 'W' or a bug 'B' or empty '0'. You can place a bomb at any empty cell with value '0'. The bomb will explose in the following four directions: up, down, left and right. The explosions will be stopped by the walls or the border of the matrix. Consequently, bugs that are covered by the explosions will be eliminated.

We use the following:

- $V_l[i][j]$ the total number of bugs when walking in the *left* direction of grid(i, j) (included) before hitting the wall.
- $V_r[i][j]$ the total number of bugs when walking in the *right* direction of grid(i, j) (included) before hitting the wall.
- $V_u[i][j]$ the total number of bugs when walking in the *uo* direction of grid(i, j) (included) before hitting the wall.
- $V_d[i][j]$ the total number of bugs when walking in the *down* direction of grid(i, j) (included) before hitting the wall.

When placing the bomb at the position (i, j) in M, we calculate the number of eliminated bugs in sum of four corresponding values, *i.e.*, $V_l[i][j]$ + $V_r[i][j]$+$V_u[i][j]$ + $V_d[i][j]$. For example, when we place bomb at "Empty" M[1][1], the number of eliminated bugs would be $V_l[1][1]$ + $V_r[1][1]$+$V_u[1][1]$ + $V_d[1][1]$ = 1+0+1+1 = 3. The four matrices V_l, V_r, V_u and V_d have the same dimensions as M.

	0	1	2	3
0	0	B	0	0
1	B	0	W	B
2	0	B	0	0

Figure 1: Illustration of M_1 with "Bug", "Wall" and "Empty"

	0	1	2	3
0	0	1	1	1
1	1	1	0	1
2	0	1	1	1

V_l for left direction

	0	1	2	3
0	1	1	0	0
1	1	0	0	1
2	1	1	0	0

V_r for right direction

	0	1	2	3
0	0	1	0	0
1	1	1	0	1
2	1	2	0	1

V_u for up direction

	0	1	2	3
0	1	2	0	1
1	1	1	0	1
2	0	1	0	0

V_d for down direction

Figure 2: Illustration of solution matrices

Task 2.1 Fill in four direction matrices below.

	0	1	2	3
0	0	0	0	B
1	0	W	0	0
2	0	0	W	0
3	B	B	0	0

Figure 3: Illustration of M_2 with "Bug", "Wall" and "Empty"

	0	1	2	3
0				
1				
2				
3				

V_l for left direction

	0	1	2	3
0				
1				
2				
3				

V_r for right direction

	0	1	2	3
0				
1				
2				
3				

V_u for up direction

	0	1	2	3
0				
1				
2				
3				

V_d for down direction

Figure 4: Illustration of solution matrices

Solution:

	0	1	2	3
0	0	0	0	1
1	0	0	0	0
2	0	0	0	0
3	1	2	2	2

V_l for left direction

	0	1	2	3
0	1	1	1	1
1	0	0	0	0
2	0	0	0	0
3	2	1	0	0

V_r for right direction

	0	1	2	3
0	0	0	0	1
1	0	0	0	1
2	0	0	0	1
3	1	1	0	1

V_u for up direction

	0	1	2	3
0	1	0	0	1
1	1	0	0	0
2	1	1	0	0
3	1	1	0	0

V_d for down direction

Figure 5: Illustration of solution matrices

Task 2.2 Write down Recursive problem formulation for $V_l[i][j]$, $V_r[i][j]$, $V_u[i][j]$ and $V_d[i][j]$ for grid M[i][j].

Solution:

$$V_l[i,j] = \begin{cases} 0 & M[i][j] = \text{"}W\text{"} \\ V_l[i,j-1]+1 & M[i][j] = \text{"}B\text{"} \\ V_l[i,j-1] & M[i][j] = \text{"}0\text{"} \end{cases}$$

$$V_r[i,j] = \begin{cases} 0 & M[i][j] = \text{"}W\text{"} \\ V_r[i,j+1]+1 & M[i][j] = \text{"}B\text{"} \\ V_r[i,j+1] & M[i][j] = \text{"}0\text{"} \end{cases}$$

$$V_u[i,j] = \begin{cases} 0 & M[i][j] = \text{"}W\text{"} \\ V_u[i-1,j]+1 & M[i][j] = \text{"}B\text{"} \\ V_u[i-1,j] & M[i][j] = \text{"}0\text{"} \end{cases}$$

$$V_d[i,j] = \begin{cases} 0 & M[i][j] = \text{"}W\text{"} \\ V_d[i+1,j]+1 & M[i][j] = \text{"}B\text{"} \\ V_d[i+1,j] & M[i][j] = \text{"}0\text{"} \end{cases}$$

Task 2.3 Implement the C funtion `calculate(M, x, y)` that returns the maximum number of elimated bugs when a bomb is placed in M. The matrix matrix M has the dimension $x \times y$.

Solution:

```
1  #include <stdio.h>
2
3  int max(int a, int b)
4  {
5  return a>b?a:b;
6  }
7  int calculate(char M[3][4],int x, int y ){
8  int V_l[3][4] = {
9        {0,0,0,0},
10       {0,0,0,0},
```

```
11      {0,0,0,0}
12  };
13  int V_r[3][4] = {
14      {0,0,0,0},
15      {0,0,0,0},
16      {0,0,0,0}
17  };
18  int V_u[3][4] = {
19      {0,0,0,0},
20      {0,0,0,0},
21      {0,0,0,0}
22  };
23  int V_d[3][4] = {
24      {0,0,0,0},
25      {0,0,0,0},
26      {0,0,0,0}
27  };
28  int res = 0;
29  for (int i = 0; i < x; ++i) {
30      for (int j = 0; j < y; ++j) {
31          int t = (j == 0 || M[i][j] == 'W') ? 0 : V_l[i][j - 1];
32          V_l[i][j] = M[i][j] == 'B' ? t + 1 : t;
33      }
34      for (int j = y - 1; j ≥0; −−j) {
35          int t = (j == y - 1 || M[i][j] == 'W') ? 0 : V_r[i][j + 1];
36          V_r[i][j] = M[i][j] == 'B' ? t + 1 : t;
37      }
38  }
39  for (int j = 0; j < y; ++j) {
40      for (int i = 0; i < x; ++i) {
41          int t = (i == 0 || M[i][j] == 'W') ? 0 : V_u[i - 1][j];
42          V_u[i][j] = M[i][j] == 'B' ? t + 1 : t;
43      }
44      for (int i = x - 1; i ≥0; −−i) {
45          int t = (i == x - 1 || M[i][j] == 'W') ? 0 : V_d[i + 1][j];
46          V_d[i][j] = M[i][j] == 'B' ? t + 1 : t;
47      }
48  }
49  for (int i = 0; i < x; ++i) {
50      for (int j = 0; j < y; ++j) {
51          if (M[i][j] == '0') {
52              res = max(res, V_l[i][j] + V_r[i][j] + V_u[i][j] + V_d[i][j]);
53          }
54      }
55  }
56  return res;}
57  int main() {
58      char M[3][4] = {
59      {'0','B','0','0'},
60      {'B','0','W','B'},
61      {'0','B','0','0'}
62  };
63
64    int res = calculate(M,3,4);
```

```
65    printf("%d ", res);
66 }
```

Task 3

A parentheses string contains only left and right parentheses. A paretheses string is balanced if and only if left and right parentheses are correctly matched. For example, the strings "(())" and "()()" are balanced while "())(()" is not. Given a parentheses string $S[0...n-1]$ with n parentheses, determine the length of **longest** balanced subsequence of string S. The **subsequence** is derived by selecting/not selecting elements, without changing the order of the selected elements. For example, for the paretheses string "())(()", the length of longest balanced subsequence is 4. We choose the 1st, 2nd, 5th, and 6th characters to form an balanced subsequence "()()".

Let dp be a 2-D matrix with the dimension $(n+1)\times(n+1)$ for the paretheses string $S[0...n-1]$, and dp[i,j] be the length of longest balanced subsequence of subarray $S[i...j]$.

	0	1	2	3
0	0	0	2	4
1	0	0	2	2
2	0	0	0	0
3	0	0	0	0

Matrix dp' for string "(())"

	0	1	2	3
0	0	2	2	4
1	0	0	0	2
2	0	0	0	4
3	0	0	0	0

Matrix dp'' for sting "()()"

Figure 6: Illustration of solution matrix D

Task 3.1 Fill in dp''' for string "((()))".

	0	1	2	3	4	5
0	0					
1	0	0				
2	0	0	0			
3	0	0	0	0		
4	0	0	0	0	0	
5	0	0	0	0	0	0

Figure 7: Matrix dp''' for sting "((()))"

Solution:

	0	1	2	3	4	5
0	0	0	0	2	4	6
1	0	0	0	2	4	4
2	0	0	0	2	2	2
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

Figure 8: Matrix dp''' for sting "((()))"

Task 3.2 Write down the recursive problem formulation for $D[i, j]$.

Solution:

$$D[i,j] = \begin{cases} \max\{D[i+1,j-1]+2, \max\limits_{i\le k<j}\{D[i,k]+D[k+1,j]\}\} & \texttt{S[i]="(" and S[j]=")"} \\ \max\limits_{i\le k<j}\{D[i,k]+D[k+1,j]\} & \texttt{else} \end{cases}$$

Task 3.3 Write C function `calculate(S, n)` that return the length of the longest balanced subsequence of S.

```
#include <stdio.h>
#include <string.h>

int D[20][20];
int inf = 999;
int max(int a, int b)
{
return a>b?a:b;
}

int DAlgo(char S[20],int i, int j) {
    if (D[i][j] != inf){return D[i][j];
    }else{

    int res;
    if (i ≥j){
        D[i][j]=0;
        return 0;
    }
  if (S[i] == 40 && S[j] == 41) {
   int item1 = DAlgo(S,i+1,j−1) + 2;
   int item2 = DAlgo(S,i,i) + DAlgo(S,i+1,j);
    for (int k = i+1; k < j; k++) {
        item2 = max(item2,DAlgo(S,i,k) + DAlgo(S,k+1,j));
    }
   res = max(item1,item2);

   }
  else {
   int item2 = DAlgo(S,i,i) + DAlgo(S,i+1,j);
    for (int k = i+1; k < j; k++) {
```

```
32            item2 = max(item2,DAlgo(S,i,k) + DAlgo(S,k+1,j));
33        }
34        res = item2;
35    }
36    D[i][j] = res;
37    return res;}
38 }
39 int calculate(char S[20], int n){
40 return DAlgo(S,0,n-1);
41 }
42
43 int main() {
44    char S[20] = "((()))";
45    int n = strlen(S);
46    for (int i = 0; i < n; i++) {for (int j = 0; j < n; j++) {D[i][j]=inf;}}
47    int res = calculate(S,n);
48    printf("%d ", res);
49 }
```