## Dynamic Programming [DP]

- One of the easiest and most useful problem solving techniques
- Usually used to solve **Optimization** problems
- The solution of a problem is made up of the solutions of its sub-problems.
- Usually, the structure of the problem can be characterized by a recurrence equation.
- A little similar to the **Divide and Conquer [DC]** technique, except a DC algorithm usually has a *merge* step. The *merge* step is used to merge solutions to subproblems together. In DP, solutions to subproblems are used to help in the decision making process of the problem.



## Problem: Fibonacci Number

Fibonacci number was used to model the growth of rabbit populations by the Italian Mathematician, Fibonacci. Fibonacci conjectured that the number of pairs of rabbits born in a given year is equal to the number of rabbits born in each of the two previous years. The conjecture can be captured by the following recurrence relation:

$$F_n = F_{n-1} + F_{n-2} \quad F_0 = 0, F_1 = 1$$

The first twenty fibonacci nummbers are $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181$

How to write a function to compute $F_n$?

```
static int f(int n){
  if (n<2) return n;
  else return f(n-1)+f(n-2);
}
```

What is the problem?



```
                              F[6]
                                |
                    |-----------------------|
                    |                       |
                  F[5]                    F[4]
                    |                       |
              |-----------|           |-----------|
              |           |           |           |
            F[4]        F[3]        F[3]        F[2]
              |           |           |           |
           |-----|     |-----|     |-----|     |-----|
           |     |     |     |     |     |     |     |
         F[3]  F[2]  F[2]  F[1]  F[2]  F[1]  F[1]  F[0]
           |     |     |           |
           |    (*)   (*)         (*)
        |-----|
        *     |
      F[2]  F[1]
        |
     |-----|
     |     |
   F[1]  F[0]
```



## Table Method and Memoization Method

```
// Table Method
static long f1(int n){
  long[] fib;
 if (n<2) return n;
  else{
    fib = new long[n+1];
    fib[0]=0; fib[1]=1;
    for (int i=2; i<=n; i++)
      fib[i]=fib[i-1]+fib[i-2];
    return fib[n];
  }
}

// Memoization Method
// static long[] gF=new long[n];
// gF[0]=0; gF[1]=1; gF[2...n]=-1
static long f2(int n){
  if (gF[n]>=0) return gF[n];
  else{
   gF[n]=f2(n-1)+f2(n-2);
   return gF[n];
  }
}
```

## Chain Matrix Multiplication

- Suppose you are to multiply 4 matrices, $A, B, C, D$ of dimensions $40 \times 20$, $20 \times 400$, $400 \times 10$, $10 \times 100$.
- Multiplying a $m \times n$ matrix and a $n \times p$ matrix requires $mnp$ multiplications.
- $(((A \times B) \times C) \times D)$ needs $40.20.400 + 40.400.10 + 40.10.100 = 520,000$ multiplications
- $((A \times ((B \times C) \times D))$ needs $20.400.10 + 20.10.100 + 40.20.100 = 180,000$ multiplications
- $((A \times (B \times C)) \times D)$ needs $20.400.10 + 40.20.10 + 40.10.100 = 128,000$ multiplications

How to find the best sequence of multiplications? Examining all possible sequence will take $O(2^n)$. In fact there are $C(n)$ of such combinations, where $C(n)$ called the *Catalan* number



## Use Dynamic Programming!

Suppose that $M = m_1 m_2 \ldots m_q$
where $m_i$ is a $r_i \times r_{i+1}$ matrix, $1 \leq i \leq q$
Let $M_{ij} = m_i \times \ldots \times m_j$, where $i \leq j$
Let $c_{ij}$ be the cost of the optimal way to compute $M_{ij}$
Let $d_{ij}$ be the position, $k$, of the optimal computation of $M_{ij}$ where $M_{ij} = M_{ik} \times M_{k+1j}$

### What is the recurrence relation?

$C(i,i) = 0, \quad 1 \leq i \leq q$
$C(i,i+1) = r_i.r_{i+1}.r_{i+2}, \quad d_{ij} = i, \quad 1 \leq i \leq q$

$C(i,i+s) =$
$\min_{i \leq k < i+s}\{C(i,k) + C(k+1,i+s) + r_i.r_{k+1}.r_{i+s+1}\}$

where $1 \leq i \leq q-s, \quad 1 < s < q, \quad d_{ij} = k$ which results in the minimum $C(i,i+s)$



## An Example

$q = 4, \ (r_1, r_2, r_3, r_4) = (10, 5, 1, 10, 2)$

$C(1,4) = \min\{C(1,1) + C(2,4) + 100, C(1,2) + C(3,4) + 20, C(1,3) + C(4,4) + 200\}$

$C(1,1) = 0, C(1,2) = 50, C(3,4) = 20, C(4,4) = 0$

$C(2,4) = \min\{C(2,2) + C(3,4) + 10, C(2,3) + C(4,4) + 100\}$

$C(2,4) = \min\{0 + 20 + 10, 50 + 0 + 100\} = 30$

$C(1,3) = \min\{C(1,1) + C(2,3) + 500, C(1,2) + C(3,3) + 100\}$

$C(1,3) = \min\{0 + 50 + 500, 50 + 0 + 100\} = 150$

$C(1,4) = \min\{C(1,1) + C(2,4) + 100, C(1,2) + C(3,4) + 20, C(1,3) + C(4,4) + 200\}$

$C(1,4) = \min\{0 + 30 + 100, 50 + 20 + 20, 150 + 0 + 200\} = 90$



## Solution: Table Method

```
static int[][] c = new int[100][100];
static int[][] d = new int[100][100];
static int[]   r = new int[101];

static int optimal(int q){
  for (int i=1; i<=q; i++){
      c[i][i]=0;
      c[i][i+1]= r[i]*r[i+1]*r[i+2];
  }
  for (int s=2; s<q ; s++)
    for (int i=1; i<=q-s; i++){
       int mymin=0x7FFFFFFF;
       for (int k=i; k<i+s; k++){
          int temp=c[i][k]+c[k+1][i+s]+
                   r[i]*r[k+1]*r[i+s+1];
          if (temp<mymin){
             mymin=temp; d[i][i+s]=k;
          }
       }
       c[i][i+s]=mymin;
   }
  return c[1][q];
}
```

## Solution: Memoization Method

```
static int[][] c = new int[100][100];
static int[][] d = new int[100][100];
static int[]   r = new int[101];

static int C(int i, int j){
  if (c[i][j]>0) //already computed
     return c[i][j];
  //not computed, compute now!
  if (i==j) return 0; //one matrix
  if (i==j-1){ //two matrices
     c[i][j]=r[i]*r[i+1]*r[i+2];
     d[i][j]=i;
     return c[i][j];
  }
  //more than two matrices
  int u=C(i,i)+C(i+1,j)+r[i]*r[i+1]*r[j+1];
  d[i][j]=i;
  //compute the remaining min terms
  for (int k=i+1; k<j; k++){
      int t=C(i,k)+C(k+1,j)+r[i]*r[k+1]*r[j+1];
      if (t<u){//smaller term
         u=t; d[i][j]=k;
      }
  }
  c[i][j]=u;
  return u;
}
```



## String Reconstruction

- Suppose that all blanks and punctation marks have been accidentally removed from a text file, and its beginning was corrupted with a few extra characters. For example, "xytobeornottobe". You want to reconstruct the file using a dictionary.
- Can be solved by using a dynamic programming algorithm.
- What is the recurrence relation?
- Let $f[1 \dots n]$ be the file of characters. Let $T[i,j]$ be a 2-D boolean already. $T[i,j]$ is true iff the string $f[i \dots j]$ is the concaternation of words from the dictionary.
- The recurrence relation is:

$$T[i,j] = dict(f[i \dots j]) \vee \bigvee_{i \le k < j} (T[i,k] \wedge T[k+1,j])$$



## Solution: Table Method

```
//T[i][j] is initialized to false
 for (int d=1; d<n; d++)
     for (int i=1; i<=n-d; i++){
          j=i+d;
          if (dict(x(i,j)))
             T[i][j]=true;
          else{
             for (int k=i; k<j; k++)
                  if (T[i][k] && T[k+1][j])
                     T[i][j]=true;
          }
     }
```



## Optimal Binary Search Tree

Suppose that you know the frequency which keywords appear in programs in pascal. For example:

| | |
|---|---|
| begin | 5% |
| do | 40% |
| else | 8% |
| end | 4% |
| if | 10% |
| then | 10% |
| while | 23% |

We want to organize them in a binary search tree such that the keyword in the root is lexicographically bigger than any keywords in its left subtree and smaller than all keywords in its right subtree as shown:

```
                 end
                  |
       |-----------------|
       |                 |
      do               then
       |                 |
  |---------|      |----------|
  |         |      |          |
begin     else     if        while
```

The cost of this tree is 2.42. This means that the average keyword finds its position after 2.42

comparisons. However the optimal binary search tree uses 2.18 comparisons. This tree is shown below:

```
            do
             |
     |-----------------|
     |                 |
   begin             while
                       |
             |---------
             |
            if
             |
     |-----------------|
     |                 |
   else              then
     |
    end
```

What is the recurrence relation?

Let $p_i$ be the probability of the $i$th keyword. We define $P_{ij} = \sum_{k=i}^{j} p_k$. Let $T(i,j)$ be the average number of comparisons in the optimal tree for keywords $i$ through $j$. The recurrence relation is:

$$T(i,j) = \min_{i \le r \le j} [T(i,r-1) + T(r+1,j) + P_{ij}]$$

## Largest Independent Set of A Tree

Almost all NP-Complete problems are easy to solve on trees. The maximum independent set problem requires you to find as many nodes as possible such that there is no edge between any two nodes.

What is the recurrence relation?

If the parent of $j$ is $i$, then $p(j) = i$ is true.

$$I(i) = \max\{\sum_{p(j)=i} I(j), 1 + \sum_{p(p(j))=i} I(j)\}$$



## Partition Problem

Given an arrangement $S$ of non negative numbers $\{s_1, \ldots, s_n\}$ and an integer $k$. Partition $S$ into $k$ ranges so as to minimize the maximum sum over all ranges.

```
100 100 100 | 100 100 100 | 100 100 100
|---300-----|-----300-----|-----300----|


100 200 300 | 400 500 600 | 700 800 900
|---600-----|----1500-----|----2400----|


100 200 300 400 500 | 600 700 |  800 900
|-------1500--------|---1300--|---1700--|
```

What is the recurrence relation?

Let $M[n,k]$ be optimal cost of the partitioning the set of non negative numbers $\{s_1, \ldots, s_n\}$ into $k$ ranges.

$M[n,k] = \min_{1 \le i \le n} \max(M[i,k-1], \sum_{j=i+1}^{n} s_j)$
$M[1,k] = s_1$
$M[n,1] = \sum_{i=1}^{n} s_i$



## Longest Increasing Subsequence

Given a sequence of numbers $S = \{s_1, \ldots, s_n\}$ what is the longest increasing subsequence?

```
S = (9,5,2,8,7,3,1,6,4)

Answer is 2,3,6 or 2,3,4
```

What is the recurrence relation?

$l[i] = \max_{0<j<i} l[j] + 1$ if $s_j < s_i$
$l[0] = 0$

## 0/1 Knapsack Problem

$z = \max \sum_{j=1}^{n} c_j x_j$
$\sum_{j=1}^{n} a_j x_j \leq b$
$x \in \{0, 1\}$

where the coefficients $\{a_j\}_{j=1}^{n}$, $\{c_j\}_{j=1}^{n}$ and $b$ are positive integers.

Solution?

Let $F[i, y]$ be the value of an optimal solution with remaining capacity $y$ and remaining objects $i, i+1, \ldots, n$.

```
int F(int i, int y){
   if (i==n) return (y<a[n])? 0 : c[n];
   if (y<a[i]) return F(i+1,y);
   else return max(F(i+1,y), F(i+1,y-a[i])+c[i]);
}
```

Problem?
Try $n = 5$, $c = [6, 3, 5, 4, 6]$, $a = [2, 2, 6, 5, 4]$

Can you rewrite the code using the table method, the memoization technique and the tuple method?



## Maximum Sum

Given a 2-dimensional (Max 100x100) array of positive and negative integers, find the sub-rectangle with the largest sum. The sum of a rectangle is the sum of all the elements in that rectangle. In this problem the sub-rectangle with the largest sum is referred to as the maximal sub-rectangle. A sub-rectangle is any contiguous sub-array of size 1x1 or greater located within the whole array.



## An Example

The maximal sub-rectangle of the array:

| | | | |
|---|---|---|---|
| 0 | -2 | -7 | 0 |
| 9 | 2 | -6 | 2 |
| -4 | 1 | -4 | 1 |
| -1 | 8 | 0 | -2 |

is in the lower-left-hand corner:

| | |
|---|---|
| 9 | 2 |
| -4 | 1 |
| -1 | 8 |

and has the sum of 15.



## Brute Force Method

```
static int BruteForce(int x, int y){
  int max=0;int s=0;
  for(int si=1;si<=x;si++){
    for(int sj=1;sj<=y;sj++){
      for(int ei=si;ei<=x;ei++){
        for(int ej=sj;ej<=y;ej++{
          for(int ci=si;ci<=ei;ci++){
            for(int cj=sj;cj<=ej;cj++){
              s+=square[ci][cj];
            }
          }
          if(s>max)
             max=s;
        }
      }
    }
  }
  return max;
}
```

Worst case time: $100^6 = 1000000 * 10^6$, Not doable!

## Dynamic Programming

How to save time from repeated computation?

First, get a new 2-dimensional array, each entry (x,y) is the sum of the sub-square (0,0,x,y) in the given array:

```
 for(int i=1; i<s; i++){
   for(int j=1; j<s; j++){
     sum[i][j]=array[i][j]+sum[i][j-1]
               +sum[i-1][j]-sum[i-1][j-1];
   }
 }
```



Then, to compute the sum of the sub-square (sx,sy,ex,ey), simply compute:

```
sum[ex][ey]-sum[sx-1][ey]
  -sum[ex][sy-1]+sum[sx-1][sy-1]
```

Why?

| | | | |
|---|---|---|---|
| 0 | -2 | -7 | 0 |
| 9 | 2 | -6 | 2 |
| -4 | 1 | -4 | 1 |
| -1 | 8 | 0 | -2 |



## Modified Program

```
static int BruteForce(int x, int y){
  int max=0;int s=0;
  for(int si=1;si<=x;si++){
    for(int sj=1;sj<=y;sj++){
      for(int ei=si;ei<=x;ei++){
        for(int ej=sj;ej<=y;ej++{
          if((s=sum[ei][ej]-sum[si-1][ej]
              -sum[ei][sj-1]+sum[si-1][sj-1])>max)
             max=s;
        }
      }
    }
  }
  return max;
}
```

Eliminate 2 for loops! Worst case time: $100^4 = 100 * 10^6$, Finish in a few seconds!