Dynamic Programming

Md. Shakil Ahmed

Software Engineer

Astha it research & consultancy ltd.

Dhaka, Bangladesh

Dynamic Programming

- Algorithmic technique that systematically <u>records</u> the answers to sub-problems in a table and <u>re-uses</u> those recorded results (rather than re-computing them).
- **Simple Example**: Calculating the Nth Fibonacci number.

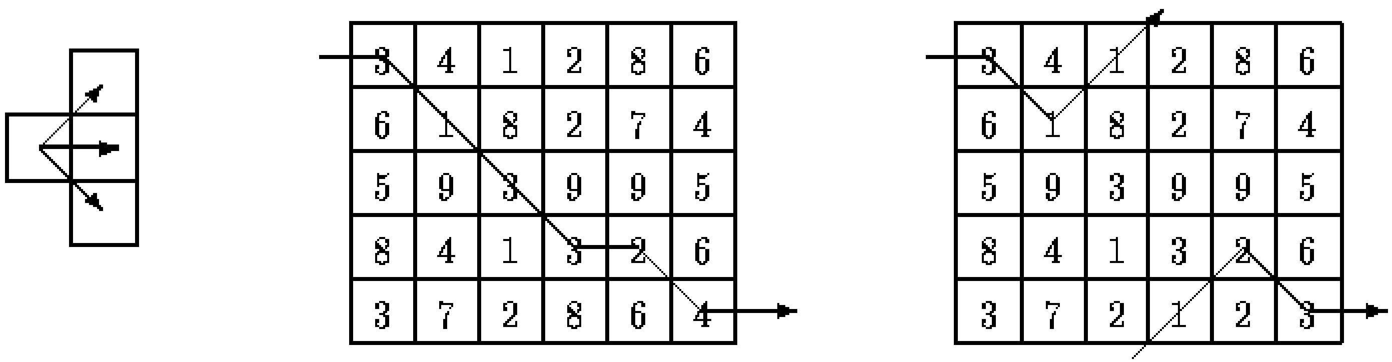
$$Fib(N) = Fib(N-1) + Fib(N-2)$$

Introduction

Topic Focus:

- Unidirectional TSP
- Coin Change
- LCS
- LIS
- The Partition Problem
- String Partition
- TSP (Bitmask)
- Matrix-chain multiplication

Unidirectional TSP



Output minimal-weight path, and the second line is the cost of a minimal path.

Input:

5 6

3 4 1 2 8 6

6 1 8 2 7 4

5 9 3 9 9 5

8 4 1 3 2 6

3 7 2 8 6 4

Output:

1 2 3 4 4 5

16

Source Code

```
fix= infinity;
for(i=0;i<n;i++)
 for(j=0;j<m;j++)
 {
   scanf("%lld",&sa[i][j]);
   si[i][j]=fix;
   t1[i][j]=-1;
 }
 for(i=0;i<n;i++)
 si[i][m-1]=sa[i][m-1];
```

```
for(j=m-1;j>0;j--)
 for(i=0;i<n;i++)
 {
  if(si[i][j-1] > si[i][j] + sa[i][j-1] || (si[i][j-1] ==
si[i][j] + sa[i][j-1] && t1[i][j-1] > i))
  {
            si[i][j-1]=si[i][j]+sa[i][j-1];
            t1[i][j-1]=i;
   }
  k=i-1;
  if(k<0)
  k=k+n;
  if(si[k][j-1] > si[i][j] + sa[k][j-1]  || (si[k][j-1] ==
si[i][j] + sa[k][j-1] && t1[k][j-1] > i))
  {
            si[k][j-1]=si[i][j]+sa[k][j-1];
            t1[k][j-1]=i;
}
```

Source Code

```
k=i+1;
k=k%n;
 if(si[k][j-1] > si[i][j] + sa[k][j-1] ||
    (si[k][j-1] == si[i][j] + sa[k][j-1]
    && t1[k][j-1] > i))
 {
    si[k][j-1]=si[i][j]+sa[k][j-1];
    t1[k][j-1]=i;
  }
 }
min=fix;
 for(i=0;i<n;i++)
 if(si[i][0]<min)
 {
min=si[i][0];
k=i;
}
```

```
printf("%ld",k+1);

 j=0;

 while(t1[k][j]!=-1)
 {
  printf(" %ld",t1[k][j]+1);
  k=t1[k][j];
  j++;
 }

 printf("\n%lld\n",min);
```

Sample Problems:

•UVA Online Judge: 116.

Coin change

- **Coin change** is the problem of finding the number of ways to make **change** for a target amount given a set of denominations. It is assumed that there is an unlimited supply of **coin**s for each denomination. An example will be finding **change** for target amount 4 using **change** of 1,2,3 for which the solutions are (1,1,1,1), (2,2), (1,1,2), (1,3).

Coin Change Source Code

```
long s[6], sa[7600]={0},
   i, j, k;

s[0]=1;
s[1]=5;
s[2]=10;
s[3]=25;
s[4]=50;

sa[0]=1;
```

```
for(j=0;j<=4;j++)
{
  for(i=0;i<=7500;i++)
  {
      if(i+s[j]>7500)
      break;

      sa[i+s[j]]+=sa[i];
  }
 }

while(scanf("%ld",&i)==1)
{
    printf("%ld\n",sa[i]);
}
```

Coin change

- Sample Problems:

1. UVA Online Judge: 147, 166, 357, 674, 10306, 10313, 11137, 11517.

The longest common subsequence (LCS) problem

- A string : A = b a c a d
- A subsequence of A: deleting 0 or more symbols from A (not necessarily consecutive).

e.g. ad, ac, bac, acad, bacad, bcd.

- Common subsequences of A = b a c a d and

  B = a c c b a d c b : ad, ac, bac, acad.

- The longest common subsequence (LCS) of A and B:

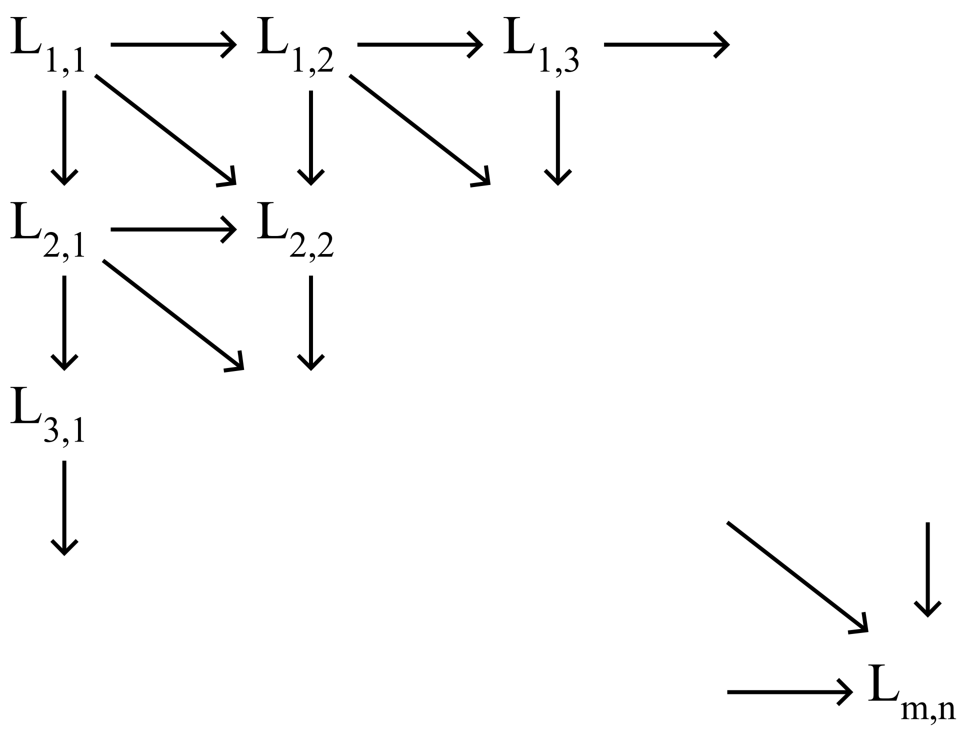
  a c a d.

The LCS algorithm

- Let $A = a_1\ a_2 \ldots a_m$ and $B = b_1\ b_2 \ldots b_n$
- Let $L_{i,j}$ denote the length of the longest common subsequence of $a_1\ a_2 \ldots a_i$ and $b_1\ b_2 \ldots b_{j.}$
- $$L_{i,j} = \begin{cases} L_{i-1,j-1} + 1 & \text{if } a_i = b_j \\ \max\{ L_{i-1,j}, L_{i,j-1} \} & \text{if } a_i \neq b_j \end{cases}$$

  $L_{0,0} = L_{0,j} = L_{i,0} = 0$ for $1 \le i \le m$, $1 \le j \le n$.

- Time complexity: O(mn)
- The dynamic programming approach for solving the LCS problem:



Tracing back in the LCS algorithm

- e.g. A = b a c a d, B = a c c b a d c b

						B				
			a	c	c	b	a	d	c	b
		0	0	0	0	0	0	0	0	0
	b	0	0	0	0	1	1	1	1	1
	a	0	1	1	1	1	2	2	2	2
A	c	0	1	2	2	2	2	2	3	3
	a	0	1	2	2	2	3	3	3	3
	d	0	1	2	2	2	3	4	4	4

- After all $L_{i,j}$'s have been found, we can trace back to find the longest common subsequence of A and B.

LCS Source Code

```
void LCS(){
 for(i=0;i<=m;i++)
     for(j=0;j<=n;j++)
     {
          c[i][j]=0;
          b[i][j]='0';
      }
 for(i=1;i<=m;i++)
     for(j=1;j<=n;j++)
     {
          if(x[i]==y[j])
          {
               c[i][j]=c[i-1][j-1]+1;
               b[i][j]='1';
          }
          else if(c[i-1][j]>=c[i][j-1])
          {
               c[i][j]=c[i-1][j];
               b[i][j]='2';
          }
          else
          {
               c[i][j]=c[i][j-1];
               b[i][j]='3';
          }
     }
}
```

```
void printSequence(long int i,long int j)
{
    if(i==0||j==0)
    return ;
    if(b[i][j]=='1')
    {
    printSequence (i-1,j-1);
    printf("%c",x[i]);
    }
    else if(b[i][j]=='2')
        printSequence (i-1,j);
    else
        printSequence (i,j-1);
}
```

Sample Problems:

- UVA Online Judge: 10066, 10405.

Longest Increasing Subsequence

- The longest increasing subsequence problem is to find a subsequence of a given sequence in which the subsequence elements are in sorted order, lowest to highest, and in which the subsequence is as long as possible. This subsequence is not necessarily contiguous.
- A simple way of finding the longest increasing subsequence is to use the Longest Common Subsequence (Dynamic Programming) algorithm.

1. Make a sorted copy of the sequence A, denoted as B. $O(n\log(n))$ time.
2. Use Longest Common Subsequence on with A and B. $O(n^2)$ time.

Simple LIS

```
function lis_length( a )
    n := a.length
    q := new Array(n)
    for k from 0 to n:
        max := 0;
        for j from 0 to k, if a[k] > a[j]:
            if q[j] > max, then set max = q[j].
        q[k] := max + 1;
    max := 0
    for i from 0 to n:
        if q[i] > max, then set max = q[i].
    return max;
```

Simple LIS

- Complexity O(n^2)

```
for(i=0;i<n;i++)
{
    scanf("%ld",&A[i]);
    Len[i]=1;
    Pre = -1;
}
Max = 0; Position = 0;
for(i=0;i<n;i++)
{
    if(Len[i]>Max)
    {
        Max = Len[i];
        Position = i;
    }
```

```
for(j=i+1;j<n;j++)
    If(A[j]>=A[i] && Len[j]<Len[i]+1)
        {
        Len[j]=Len[i]+1;
        Pre[j]=i;
        }
}
```

For sequence Print:

```
void Show(long h1)
{
        If(Pre[h1]!=-1)
                Show(Pre[h1]);
        Printf("%ld\n",A[h1]);
}

Show(Position);
```

Efficient LIS

- Complexity O(n*log(n))

```
L = 0
for i = 1, 2, ... n:
    binary search for the largest positive j ≤ L
    such that X[M[j]] < X[i] (or set j = 0 if no such value exists)
        P[i] = M[j]
    if j == L or X[i] < X[M[j+1]]:
        M[j+1] = i
    L = max(L, j+1)
```

Efficient LIS

```
long BinarySearch(long left,long right,long value)
{
long mid = (left+right)/2;
while(left<=right)
{
    if(temp[mid]==value)
            break;
    else if(temp[mid]<value)
            left = mid+1;
    else
            right = mid – 1;
    mid = (left+right)/2;
}
return mid;
}
```

Efficient LIS

```
void LIS(long N)
{   long n=0,i,mid;
    for(i=0;i<N;i++)
    {
    if(n==0)
    {       temp[n]=sa[i];tempP[n]=i;
            A[i]=1;n++;          }
    else
    {
    mid = BinarySearch(0,n-1,sa[i]);
    while(mid>0&&temp[mid]>sa[i])
            mid–;
    if(temp[mid]<sa[i])
    {
            A[i]=mid+2;P[i]=tempP[mid];
            if(mid+1>=n)
            {
            temp[n]=sa[i]; tempP[n]=i;
            n++;
            }
            else if(temp[mid+1]>sa[i])
            {
            temp[mid+1]=sa[i];
            tempP[mid+1]=i;
            }
    }
    else if(temp[0]>sa[i])
    {
    temp[0]=sa[i];
    tempP[0]=i;
    }       }       }
}
```

Efficient LIS

```
void show(long h1)
    {
    if(P[h1]==-1)
    printf(“%ld”,sa[h1]);
    else
    {
    show(P[h1]);
    printf(” %ld”,sa[h1]);
    }
    }
```

```
for(i=0;i<n;i++)
{
scanf(“%ld”,&sa[i]);
P[i]=-1;
}
LIS(n);
max = 0;
x = 0;
for(i=0;i<n;i++)
if(A[i]>max)
{
max = A[i];
x = i;
}
printf(“%ld\n”,max);
show(x);
```

LIS

- Sample Problems:

1. UVA Online Judge: 111, 231, 437, 481, 497, 1196, 10131, 10534, 11456, 11790.

The Partition Problem

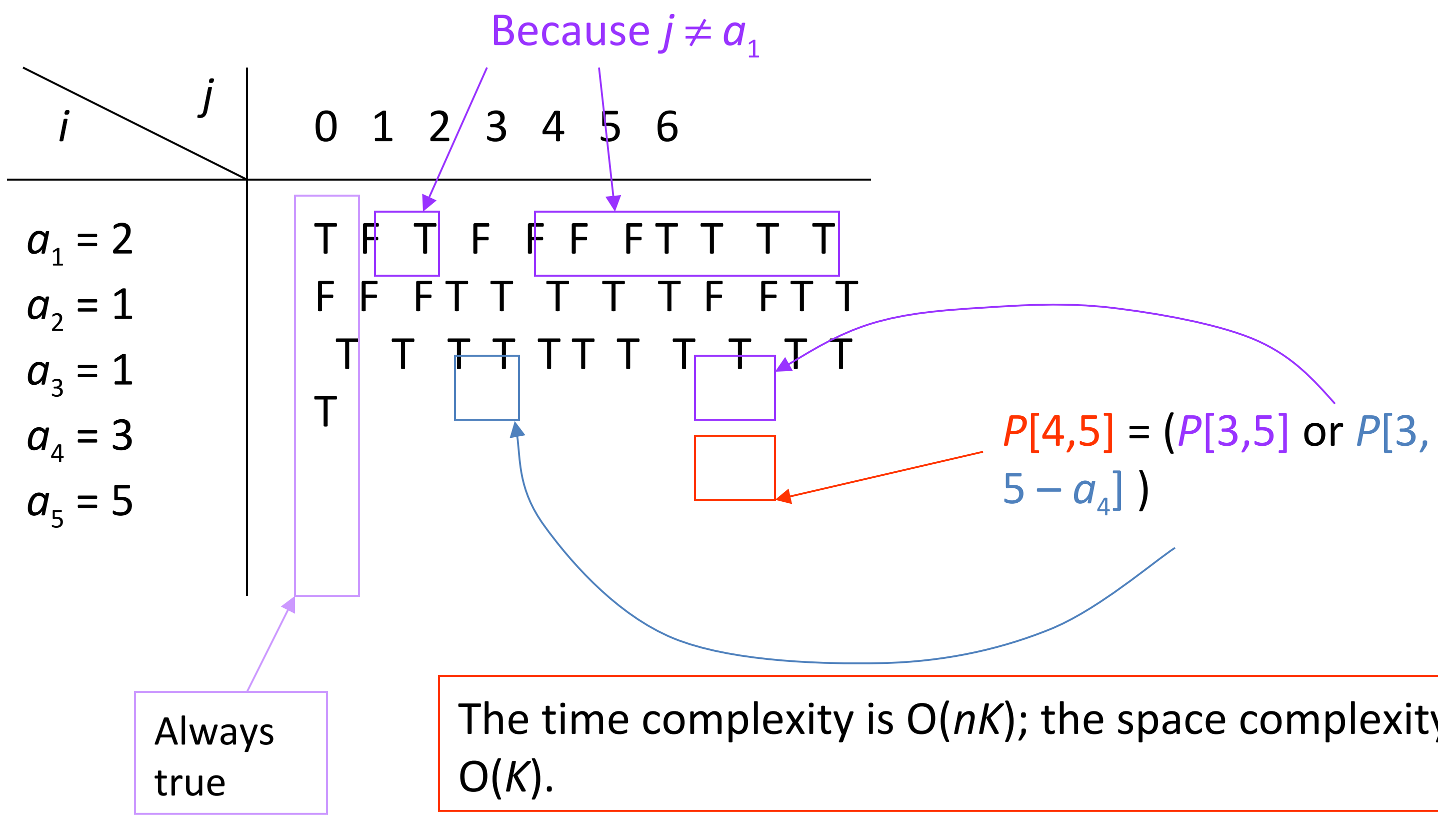
Given a set of positive integers, $A = \{a_1, a_2, \ldots, a_n\}$. The question is to select a subset B of A such that the sum of the numbers in B equals the sum of the numbers not in B, i.e., . We may assume that the sum of all numbers in A is $2K$, an even number. We now propose a dynamic programming solution. For $1 \le i \le n$ and $0 \le j \le K$,

$$\sum_{a_i \in B} a_i = \sum_{a_j \in A-B} a_j$$

define $P[i, j]$ = True if there exists a subset of the first i numbers a_1 through a_i whose sum equals j; False otherwise.

Thus, $P[i, j]$ = True if either $j = 0$ or if ($i = 1$ and $j = a_1$). When $i > 1$, we have the following recurrence: $P[i, j] = P[i-1, j]$ or ($P[i-1, j-a_i]$ if $j - a_i \ge 0$) That is, in order for $P[i, j]$ to be true, either there exists a subset of the first $i - 1$ numbers whose sum equals j, or whose sum equals $j - a_i$ (this latter case would use the solution of $P[i-1, j-a_i]$ and add the number a_i. The value $P[n, K]$ is the answer.

Example: Suppose A = {2, 1, 1, 3, 5} contains 5 positive integers. The sum of these number is 2+1+1+3+5=12, an even number. The partition problem computes the truth value of $P[5, 6]$ using a tabular approach as follows:



The time complexity is $O(nK)$; the space complexity $O(K)$.

The Partition Problem Source Code

```
sum = 0;
for(i=0;i<n;i++)
    sum += A[i];

for(i=0;i<=sum/2;i++)
    Flag[i]=0;

Flag[0]=1;

for(i=0;i<n;i++)
    for(j=sum/2-A[i];j>=0;j--)
    if(Flag[j]==1)
        Flag[j+A[i]]=1;
```

```
for(i=sum/2;i>=0;i--)
        if(Flag[i]==1)
                break;

Dif = (sum – i) – i;
```

String Partition

A string of digits instead of a list of integers. There are many ways to split a string of digits into a list of non-zero-leading (0 itself is allowed) 32-bit signed integers. What is the maximum sum of the resultant integers if the string is split appropriately? A string of at most 200 digits.

Input:

11111111111111111111111111111111111111111111111111111

Output:

5555555666

Source Code

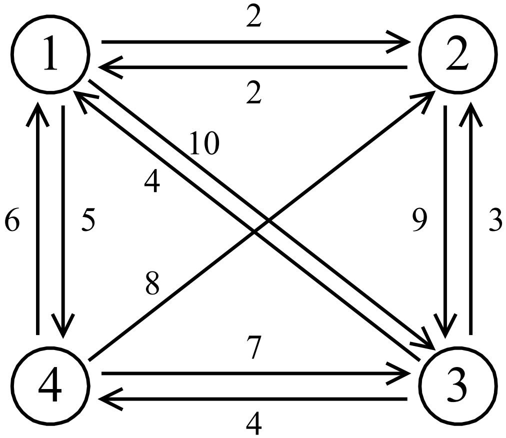
```
Limit = 2147483647;
Len = strlen(input);
for(i=0;i<=Len;i++)
    Max[i]=0;
for(i=0;i<Len;i++)
{
    Num = 0;
    for(j=i;j<Len;j++)
    {
        Num = Num * 10 + input[j]-'0';
        if(Num>Limit)
            break;
        if(Num+Max[i]>Max[j+1])
            Max[j+1] = Num+Max[i];
        if(j==i&&input[j]=='0')
            break;
    }
}

printf("%lld\n",Max[Len]);
```

Sample Problems:

- UVA Online Judge: 11258.

The traveling salesperson (TSP) problem

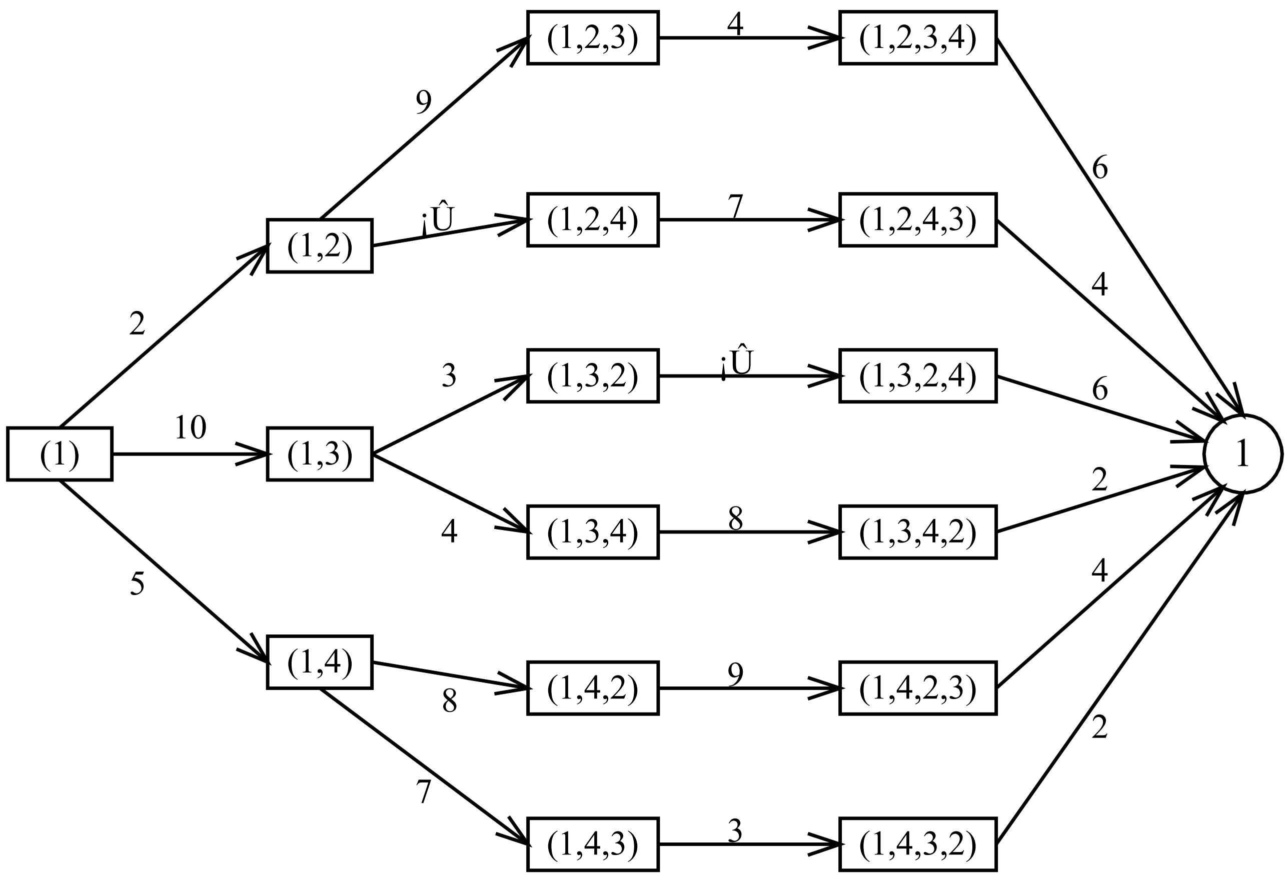
- e.g. a directed graph :



- Cost matrix:

$$
\begin{array}{c}
\\
1\\
2\\
3\\
4
\end{array}
\begin{array}{c}
\begin{array}{cccc} 1 & 2 & 3 & 4 \end{array}\\
\begin{bmatrix}
\infty & 2 & 10 & 5\\
2 & \infty & 9 & \infty\\
4 & 3 & \infty & 4\\
6 & 8 & 7 & \infty
\end{bmatrix}
\end{array}
$$

TSP



- A multistage graph can describe all possible tours of a directed graph.
- Find the shortest path:

  (1, 4, 3, 2, 1) 5+7+3+2=17

BIT MASK

- you are in node 1, and you have to visit node 2 5 & 10. Now, you have to fine the minimum cost for this operation.
- you have to visit l number of node, like l = 3. then node[] = 2, 3, 4
  & your source is 1. now you have to find the minimum cost of the tour.
- C[][] = cost matrix of the nodes.

BIT MASK Source Code

```
P[0]=1;
for(i=1;i<=15;i++ )
{
    P[i] = P[i-1]*2;
}


source = 1;


for(i=0;i<=P[l];i++)
{
    for(j=0;j<l;j++)
    {
            bit[i][j] = MAX;
    }
}
```

```
for(i=0;i<l;i++)
{
    if( C[ source ][ node[i] ]!=MAX )
    {
            bit[ P[i] ][i] = C[ source ][ node[i] ];
     }
}
for(i=0;i<P[l];i++)
{
     for(j=0;j<l;j++)
     {
            if( bit[i][j]!=MAX )
            {
                    for(k=0;k<l;k++)
                    {
                            if( (i%P[k+1])/P[k]==0 )
                            {
```

BIT MASK Source Code

```
if( C[node[j]][node[k]]!=MAX )
{
    if( bit[ i+P[k] ][k] > bit[i][j] + C[node[j]]
    [node[k]] )
    {
        bit[ i+P[k] ][k] = bit[i][j] +
    C[node[j]][node[k]];
    }
}
}
}
}
}
}
```

```
int res = MAX;

for(i=0;i<l;i++)
    if(bit[P[l]-1][i]!=MAX)
        if(res > bit[P[l]-1][i])
            res = bit[P[l]-1][i];

printf("%d\n",res);
```

Sample

- UVA Online Judge: 10651, 216, 10496, 11813, 10718.

Matrix-chain multiplication

- n matrices $A_1, A_2, \ldots, A_n$ with size

  $p_0 \times p_1, p_1 \times p_2, p_2 \times p_3, \ldots, p_{n-1} \times p_n$

  To determine the multiplication order such that # of scalar multiplications is minimized.
- To compute $A_i \times A_{i+1}$, we need $p_{i-1}p_ip_{i+1}$ scalar multiplications.

e.g. n=4, A_1: 3×5, A_2: 5×4, A_3: 4×2, A_4: 2×5

$((A_1 \times A_2) \times A_3) \times A_4$, # of scalar multiplications:

$$3 * 5 * 4 + 3 * 4 * 2 + 3 * 2 * 5 = 114$$

$(A_1 \times (A_2 \times A_3)) \times A_4$, # of scalar multiplications:

$$3 * 5 * 2 + 5 * 4 * 2 + 3 * 2 * 5 = 100$$

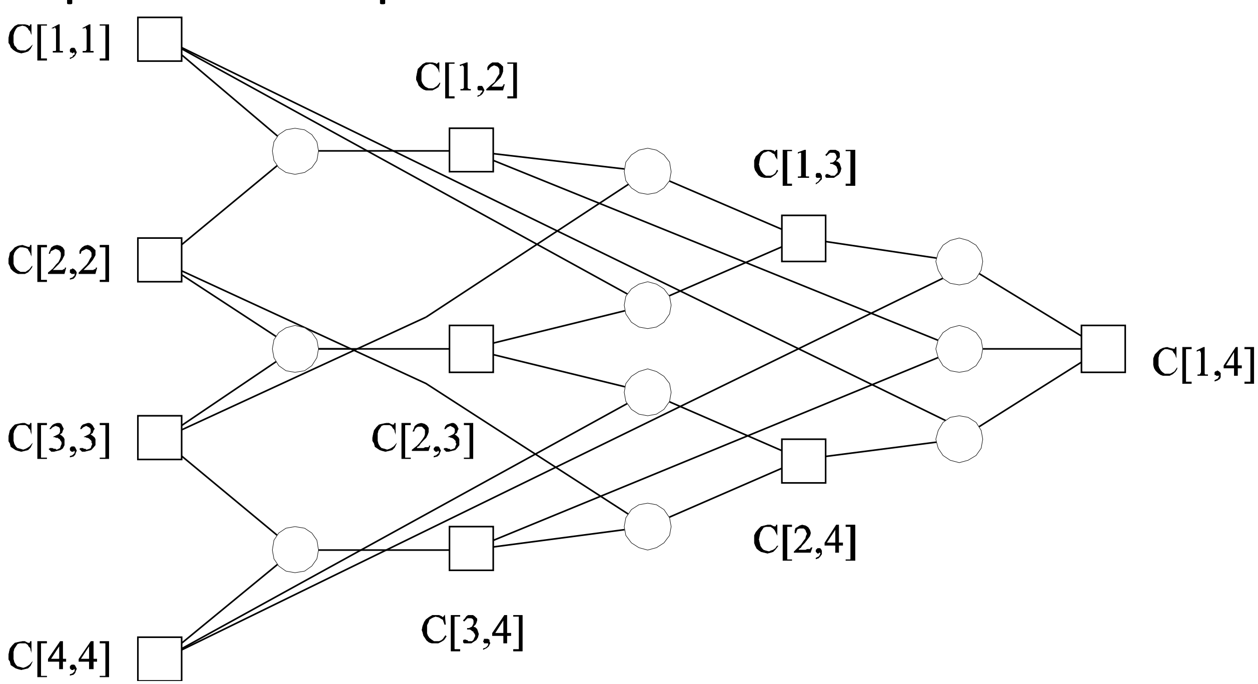
$(A_1 \times A_2) \times (A_3 \times A_4)$, # of scalar multiplications:

$$3 * 5 * 4 + 3 * 4 * 5 + 4 * 2 * 5 = 160$$

- Let m(i, j) denote the minimum cost for computing $A_i \times A_{i+1} \times \ldots \times A_j$

$$m(i,j) = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le k < j} \{ m(i,k) + m(k+1,j) + p_{i-1} p_k p_i \} & \text{if } i < j \end{cases}$$

- Time complexity : $O(n^3)$
- Computation sequence :



Matrix-chain multiplication

```
// Matrix Ai has dimension p[i-1] x p[i] for i = 1..n
Matrix-Chain-Order(int p[]) {
// length[p] = n + 1
 n = p.length - 1;
// m[i,j] = Minimum number of scalar multiplications (i.e., cost)
// needed to compute the matrix A[i]A[i+1]...A[j] = A[i..j]
// cost is zero when multiplying one matrix
for (i = 1; i <= n; i++)
    m[i,i] = 0;

for (L=2; L<=n; L++) { // L is chain length
    for (i=1; i<=n-L+1; i++) {
            j = i+L-1;
```

Matrix-chain multiplication

```
m[i,j] = MAXINT;
for (k=i; k<=j-1; k++) {
// q = cost/scalar multiplications
q = m[i,k] + m[k+1,j] + p[i-1]*p[k]*p[j];
if (q < m[i,j])
{
    m[i,j] = q;
    // s[i,j] = Second auxiliary table that stores k
    // k = Index that achieved optimal
    cost s[i,j] = k;
}
} } } }
```

Thanks!