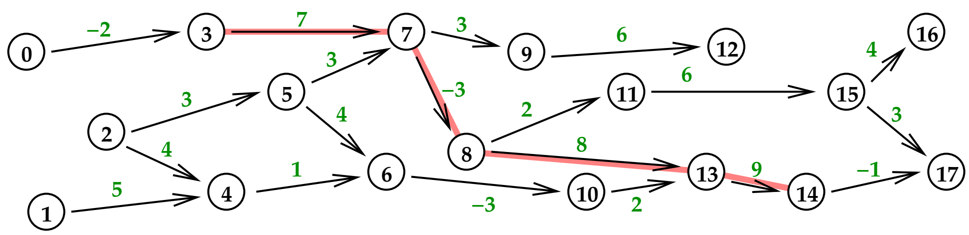
**University of Nevada, Las Vegas Computer Science 477/677 Spring 2022**

**Assignment 7: Due Tuesday May 3, 2022, midnight.**

**Name:**________________________________________________________________

1. Given an acyclic weighted directed graph $G$, write a dynamic program which finds a directed path through $G$ of maximum total weight. Let the vertices of $G$ be the integers $\{i\}_{0 \leq i < n}$ and assume there is no edge from $i$ to $j$ if $i > j$. An example of such a graph is shown in the figure below, where the maximum weight path is indicated.



There are two ways to work the problem. You only need to do one of them.

(a) Identify the subproblems.

(b) Your code should work each subprogram in topological order.

(c) Your code should print the maximal weight path.

Use whatever pseudo-code you like, but make sure it's understandable.

2. You need to store an array $A$ where $A[i][j][k]$ is defined if $0 \le k \le j \le i < N$. Note that $A$ is sparse, since the size of $A$ is $\binom{N+2}{3}$, which is roughly $N^3/6$. To save space, you store the items of $A$ in a 1-dimensional array $X[M]$ in row-major order, where $M = \binom{N+2}{3}$. You want to complete the following code.

```
int index(int i, int j, int k)
 {
  assert(i < N and j <= i and k <= j and k >= 0);
  return                                ; // Insert the index in X of A[i][j][k]
 }

fetchA(int i, int j, int k)
 {
  assert(i >= 0 and i <= j and j <= k and k < N);
  return X[index(i,j,k)]
 }
```

Hint: The formula can most easily be expressed using combinatorials, that is, entries of Pascal's triangle.

Hint: Try working out the number of predecessors for a few cases, such as $A[3][2][1]$, $A[5][3][2]$, *etc.*.

3. The number of proper divisors of a positive integer $n$ can be computed by the following C++ code.

```
int numdiv(int n)
 {
  assert(n > 0);
  int numd = 1;
  int d = 2;
  while(d*d < n)
   {
    if(n % d == 0) numd = numd+2;
    d++;
   }
  if(d*d == n) numd++;
  return numd;
 }
```

For example, numdiv(1) = 1, numdiv(2) = 1, numdiv(3) = 1, numdiv(4) = 2, numdiv(5) = 1, and numdiv(6) = 3. Note that numdiv(p) = 1 if p is prime, and that numdiv(60) = 11.

You wish to store a 2-dimensional ragged array D, where D[i][j] is the $j^{\text{th}}$ proper divisor of $i$, for all integers $i$ from 2 up to some constant $N$, in a 1-dimensional array $X$, such that D[i][j] = X[index(i,j)].

The first nine rows (for $2 \leq i \leq 10$) of D look like this:

```
1
1
1 2
1
1 2 3
1
1 2 4
1 3
1 2 5
```

which means that the first 17 entries of X are: 1 1 1 2 1 1 2 3 1 1 2 4 1 3 1 2 5

How would you implement this project?

4. For each of the following C++ code fragments: run it on your computer, observe the output, then give the asymptotic time complexity in terms of $n$. Don't hand in the output of your program.

   (a)
```
int main()
{
 int n;
 cout << "Enter n: ";
 cin >> n;
 for(int i = 1; i < n; i++)
  for(int j = 1; j < i; j = 2*j)
   cout << i << " " << j << endl;
}
```

   (b)
```
int main()
{
 int n;
 cout << "Enter n: ";
 cin >> n;
 for(int i = 1; i < n; i++)
  for(int j = i; j < n; j = 2*j)
   cout << i << " " << j << endl;
}
```

   (c)
```
int main()
{
 int n;
 cout << "Enter n: ";
 cin >> n;
 for(int i = n; i > 0; i = i/2);
  for(int j = 1; j < i; j++)
   cout << i << " " << j << endl;
}
```

   (d)
```
int main()
{
 int n;
 cout << "Enter n: ";
 cin >> n;
 for(int i = n; i > 0; i = i/2);
  for(int j = i; j < n; j++)
   cout << i << " " << j << endl;
}
```

5. Run each of the following recursive C++ code fragments on your computer and observe the ouput. Then give an asymptotic solution to the recurrence. in terms of $n$. Don't hand in the output of your program.

(a)
```
int F(int n)
 {
  if(n <= 1) return 1;
  else return 4*F(n/2)+n*n; // This is the right side of the recurrence
 }
int main()
 {
  int n;
  cout << "Enter n: ";
  cin >> n;
  cout << "F(" << n << ") = " << F(n) << endl;
 }
```

(b)
```
int F(int n)
 {
  if(n <= 1) return 1;
  else return F(3*n/5)+F(4*n/5)+1; // This is the right side of the recurrence
 }
int main()
 {
  int n;
  cout << "Enter n: ";
  cin >> n;
  cout << "F(" << n << ") = " << F(n) << endl;
 }
```

(c)
```
int F(int n)
 {
  if(n <= 1) return 1;
  else return F(sqrt(n))+1; // This is the right side of the recurrence
 }
int main()
 {
  int n;
  cout << "Enter n: ";
  cin >> n;
  cout << "F(" << n << ") = " << F(n) << endl;
 }
```

Levenshtein edit distance is used for approximate string matching. The levenshtein distance between two words $w_1$ and $w_2$ is the number of edits needed to change one to the other. Three kinds of edits are permitted.

(a) Insert a symbol.

(b) Delete a symbol.

(c) Replace a symbol with another symbol.

Find the Levenshtein distance between “abbabacaa” and “babacbacab” Show the matrix.