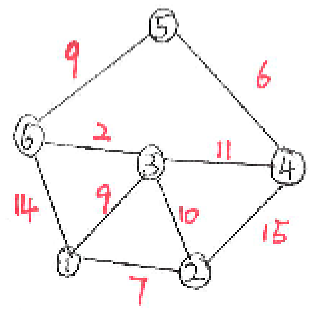
CS 323 Notes by Hui Shao /20 April 2016

CS 323 Homework 8

Hui Shao

1.

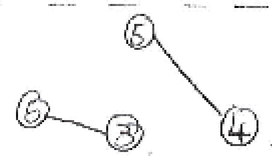


Edge	3 - 6	5 - 4	1 - 2	1 - 3	6 - 5	3 - 2	3 - 4	1 - 6	2 - 4
Weight	2	6	7	9	9	10	11	14	15

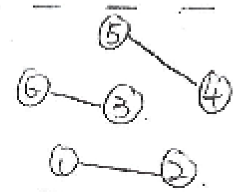
⟶ increase

Step 1: select minimum weight. is 2. ⟶

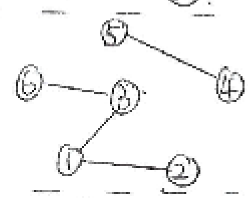
Step 2: select minimum weight of rest is 6 ⟶



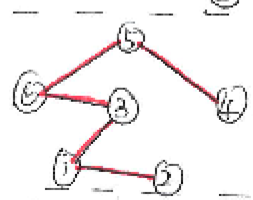
Step 3: select minimum weight of rest is 7 ⟶



Step 4: select minimum weight of rest is 9 ⟶
(There are two 9 here, select which one doesn't matter).



Step 5 select minimum weight of rest is 9 ⟶



Since we have 6 vertices, then MST will have 5 edges.
Therefore, step 5 is final result. (the same as using Prim's Algorithm).

2. Algorithm idea:

1) Construct a sum matrix S[R][C] for the given M[R][C]

   a) copy first row and first column as it is from M[ ][ ] to S[ ][ ]

   b) for other entries,use following expressions to construct S[ ][ ]

      if M[i][j] is 1 then

         S[i][j] = min (S[i][j-1], S[i-1][j], S[i-1][j-1]) +1

      else   S[i][j] = 0

2) Find the maximum entry in S[R][C]

3) Using the value and coordinates of maximum entry in S[i],print sub-matrix of M[ ][ ]

The code is Following :

```
int i , j ;
int S[R][C];
int Max_of_S, Max_i,Max_j;
for ( i = 0; i<R; i++)
    S[i][0] = M[i][0];
for ( j = 0; j<C; j++)
    S[0][j] = M[0][j];
for ( i = 1; i<R; i++) {
    for ( j=1; j<C; j++) {
        if (M[i][j] == 1)
            S[i][j] = min (S[i][j-1], S[i-1][j], S[i-1][j-1]) +1;
        else
            S[i][j] = 0;
        }
    }
}
```

3. “0-1” knapsack

for w = 0 to W

    do V[0,W] ← 0

for i = 0 to n

    do V[i,0] ← 0

    for w = 0 to W

    do if ($w_i \leq w$ and $V_i + V[i-1,w-w_i] > V[i-1][w]$)

        then $V[i,w] \leftarrow Vi + V[i-1,w-w_i]$

        else $V[i,w] \leftarrow V[i-1,w]$

4. Optimal-Triangulate

Let Minimum Cost of triangulation of vertices from i to j be minCost(i, j)

If j <= i + 2 Then

  minCost(i, j) = 0

Else

  minCost(i, j) = Min { minCost(i, k) + minCost(k, j) + cost(i, k, j) }

        Here k varies from 'i+1' to 'j-1'

Cost of a triangle formed by edges (i, j), (j, k) and (k, j) is

  cost(i, j, k) = dist(i, j) + dist(j, k) + dist(k, j)

Lecture note:

Divide And Corque

Big Integer Multiplication

Suppose x and y are two n-bit integers, and assume for convenience that n is a power of 2 (the more general case is hardly any different). As a first step toward multiplying X and Y, split each of them into their left and right halves, which are n/2 bits long:

$$x = \boxed{\;x_L\;}\boxed{\;x_R\;} = 2^{n/2}x_L + x_R$$
$$y = \boxed{\;y_L\;}\boxed{\;y_R\;} = 2^{n/2}y_L + y_R.$$

For instance, if x = 101101102 (the subscript 2 means "binary") then xL = 10112, xR = 01102, and X = 10112 × 24 + 01102. The product of x and y can then be rewritten as

$$xy = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^n\, x_L y_L + 2^{n/2}\,(x_L y_R + x_R y_L) + x_R y_R.$$

We will compute XY via the expression on the right. The additions take linear time, as do the multiplications by powers of 2 (which are merely left-shifts). The significant operations are the four n/2-bit multiplications, $x_L y_L$, $x_L y_R$,$x_R y_L$, $x_R y_R$; these we can handle by four recursive calls. Thus our method for multiplying n-bit numbers starts by making recursive calls to multiply these four pairs of n/2-bit numbers (four subproblems of half the size), and then evaluates the preceding expression in O(n) time. Writing T(n) for the overall running time on n-bit inputs, we get the recurrence relation

$$T(n) = 4T(n/2) + O(n).$$

We will soon see general strategies for solving such equations. In the meantime, this particular one works out to $O(n^2)$, the same running time as the traditional grade-school multiplication technique. So we have a radically new algorithm, but we haven't yet made any progress in efficiency. How can our method be sped up?

Although the expression for XY seems to demand four n/2-bit multiplications, as before just three will do: $x_L y_L$, $x_R y_R$, and $(x_L + x_R)(y_L + y_R)$, since $x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$. The resulting algorithm has an improved running time of

$$T(n) = 3T(n/2) + O(n).$$

The point is that now the constant factor improvement, from 4 to 3, occurs at every level of the recursion, and this compounding effect leads to a dramatically lower time bound of $O(n^{1.59})$

(https://www.cs.berkeley.edu/~vazirani/algorithms/chap2.pdf)