Illinois Institute of Technology
Department of Computer Science

Homework 7 Solutions

CS 430 Introduction to Algorithms
Spring Semester, 2014

1. Let $P[i]$ be the maximum profit from locations 1 to i.
   Charging stations should be set only k distance apart, so we get the following recurrence relation to find the maximum profit.

P[i]=max$\begin{cases} P[i] = P[j] + p[i] \text{ if } (s_i - s_j) \geq k \\ P[i-1] \text{ otherwise} \end{cases}$
.

where j is the first location after $i + k$

```
MaxProfit(S[],P[],k)
{
Declare an array Profit*1...n+ = 0 // Profit* + array denotes the maximum profit at location i
Declare Max =0 // Max contains the maximum profit of all locations less than i
for (i = 2 ; i < n; i++)
{
for(j =n; j > i ; j++)
If(S[i]-S[j] > k )

Profit[i] = Profit[i] + P[i];
}

}
```

2. A subsequence x[i]+ ... x[j] can be palindrome if x[i]=x[j] else we find maximum length palindrome substring from x[i+1]... x[j] and x[i] ... x[j-1].
   Recurrence for the maximum length palindrome substring can be defined as following:

L(i,j)= $\begin{cases} L(i+1, j-1) + 2 \text{ if } x[i] = x[j] \\ max\{L(i+1, j), L(i, j-1)\} \forall i \in (1, 2, ...n) \text{ otherwise} \end{cases}$
.

```
MaxLenPalindrome(x[1] ... x*n+)
{
for( i =1; i < n ; i++)
{
L[i,i] = 1;
}
for ( i =1; i < n ; i++)
{
```

```
for (s=1; s < n-i; s++)
{
j=s+i;
L[s,j] = calculate_len(L, x[1] ... x[n],s,j);
L[j,s] = calculate_len(L, x[1] ... x[n],j,s);
}
}
return L[1,n]
}
calculate_len recursively calculates the maximum length of the palindrome substring
calculate_len(L, x[1] ... x[n],i,j)
{
If(i == j)
Return L(i,j);

Else if (x[i] == x[j])
{
If (i+1 < j-1)
Return L(i+1,j-1)+2;
Else
Return 2;
}
Else
{
Return Max(L(i+1,j),L(i,j-1)
}
}
```

3. Recurrence for finding if change of v is possible of not can be given as below:

$$\text{D(v,i-1)}= \begin{cases} D(v-x_i, i) \vee \text{D(v, i-1)} if (\text{x}_i \leq v) \\ D(v, i-1) \text{ if } x_i < v \\ \text{false if } i = 0 \end{cases}$$

.

```
Change_possible(V,x1x2...xn)
{
Declare an array D of size V+1
D[0] = true;
For(i =1 ; i <= V ; i++)
D[i] = true;
For (v = 1 ; v < V ; v++)
{
For (j =1; j < n ++)
{
If (xj <= v)
D[v] = D[v] || D[v-xj];
Else
D[v] = false;
}
}
```

```
Return D[V] ;
}
```

4. Let $M(i, j)$ be the optimal bitonic distance between points p_i and p_j. $p_1, ..., p_n$ are sorted as per the x-axis co-ordinate. $p_{\bar{k}} p_l$ is the distance between p_j and p_k.
Recurrence relation:

$$\forall i, j s.t 1 \leq j < i \leq n M(i, j) = \begin{cases} M(i-1, j) + p_{l-\bar{1}} p_l \text{ if } (1 \leq j \leq i-2) \\ min_{1 \leq k \leq i-2}(M(i-1, k) + p_{\bar{k}} p_l) \text{ if } (j = i-1) \end{cases}$$
.

Time complexity :
Time Complexity of finding Optimal Bitonic Path = Time complexity of sorting the points + Time complexity to compute matrix $M(i, j)$ + Time complexity of find the optimal bitonic path.

Sorting n points takes $O(nlogn)$ time.
the matrix time complexity is $O(n^2)$ because there are n^2 elements in the matrix and computing each element takes $O(1)$ complexity.
To select the optimal path we look $n-1$ values of j and add distance to it to get final result which takes$O(n)$ time. So we get,
$T(n) = O(nlogn) + O(n^2) + O(n)$ Therefore, time complexity is $O(n^2)$.