Problems

Jiankai Sun

October 22, 2014

Question 1. Given n intervals (s_i, f_i) where $1 \leq i \leq n$, s_i and f_i is $i-th$ interval's start and finish time, its duration time is $f_i - s_i$. Use any algorithm to get the maximize the total sum of mutually disjoint intervals.

1. we can use dynamic programming to solve this problem
2. for n intervals (s_i, f_i), it's sorted so that $f_1 \leq f_2 \leq ... \leq f_n$
3. We use $T(i)$ to represent the maximize total sum of intervals of $((s_1, f_1), (s_2, f_2), ...(s_i, f_i))$. each interval has two choices: added to the select set or not.

   Define $p(i)$=the largest$j < i$ that interval i doesn't overlap with j.

   So the recurrence function is: $T(i) = max \begin{cases} T(p(i)) + f_i - s_i \\ T(i-1); otherwise \end{cases}$
4. boundary condition $T(0) = 0$
5. Implement the algorithm

Algorithm 1 function maxSumInterval(i)

Sort the interval according to f_i, so that $f_1 \leq f_2 \leq ... \leq f_n$.
Start time array S=$(0, s_1, s_2, ..., s_n)$, finish time array F=$(0, f_1, f_2, ..., f_n)$.

```
global T[0,1,2,...,n], T[0]=0; P[1,...,n]

for i ← 1 to n do
    if T(p(i))+f_i-s_i¿T(i-1) then
        T[i]=T(p(i))+f_i-s_i
        P[i]=p(i)
    else
        T[i]=T[i-1]
        P[i]=-1
Return T[n]
```

6. we use $P[1, 2, ..., n]$ to print selected intervals

Algorithm 2 function printInterval()

```
global P[1, ..., n]
i=n
while P[i] ≠ 0 do
    if P[i] > 0 then
        print i
        i=P[i]
print i
```

7. We use $O(nlog(n))$ to sort. For each i we need $O(n)$ to scan all the interval list and repeat n times, So the time complexity is $O(n^2)$

8. source code: max_selected_intervals.cpp

Question 2. Let $A = a_1a_2...a_m$ and $B = b_1b_2...b_n$ be two strings of characers. we want to transform A into B using following operations:

delete a character

add a character

change a character

write a dynamic programming algorithm that finds the minimum number of operations needed to transform Ainto B

1. Let F[i,j] denotes the minimum number of operations needed to transform $a_1a_2...a_i$ to $b_1b_2...b_j$.

2. Recurrence relation

$$F[i,j] = or \begin{cases} F[i-1, j-1]; if a_i = b_j \\ min \begin{cases} F[i-1, j-1] + 1; change \\ F[i, j-1] + 1; add \\ F[i-1, j] + 1; delete \end{cases} & otherwise \end{cases}$$

3. Boundary conditions

   $F[0, k] = k$ for k in [0,n]

   $F[k, 0] = k$ for k in [0,m]

4. implement the non recursive algorithm, see algorithm **??**

Algorithm 3 function MinNumOper()

```
global F[1...m; 1..n], A[1...m], B[1...n]
Initialize F[0,k] = k for k in [0,n] and F[k,0] = k for k in [0,m]
for i ← 1 to m do
    for j ← 1 to n do
        if A[i]==B[j] then F[i,j]=F[i-1,j-1]
        else
```

$$F[i,j] = min \begin{cases} F[i-1,j-1]+1; change \\ F[i,j-1]+1; add \\ F[i-1,j]+1; delete \end{cases}$$

```
return F[m,n]
```

5. in the main function, we can call the function $MinNumOper()$ to to find the minimum number of operations needed to transform A into B

6. time complexity of this algorithm is $\Theta(mn)$

7. source code: min_operation_dp.cpp