DDA 6050: Homework #2

Due on Oct 28, 2021

Professor Yixiang Fang

Haoyu Kang

Problem 1

LCS

The state transition equation is as below:

$$c[i,j] = \begin{cases} c[i-1,j-1]+1, & x[i]=y[j] \\ max(c[i-1,j],c[i,j-1]), & otherwise. \end{cases} \tag{1}$$

For this question, we adopt method of state compression instead of a traditional approach that takes $O(n^2)$ memory. In details, we use **scrolling array** which is a one-demision array to save each states. For each inside loop, $dp[j]$ represents length of LCS between i-length of prefix A and the j-length of prefix B. The variable *temp* will save the value of LCS between i-1 prefix of A and the j-1 prefix of B.

Hence the optimal method of LCS just takes $O(n)$ space complexity and $O(n^2)$ time complexity.
The cpp code is attached as follow:

```cpp
#include<iostream>
#include<vector>
#include<cstdio>
using namespace std;
int main(){
    int num;
    vector<int> seq1,seq2;
    while(scanf("%d",&num)) {
        seq1.push_back(num);
        if(cin.get()=='\n') break;
    }
    while(scanf("%d",&num)){
        seq2.push_back(num);
        if(cin.get()=='\n') break;
    }
    int n=seq1.size();
    vector<int> dp(n+1,0);
    int pre;
    for(int i=1;i<n+1;i++){
        for(int j=1;j<n+1;j++){
            int temp=dp[j];
            if(i==1 && j==1) dp[j]=seq1[i-1]==seq2[j-1]?1:0;
            else if(i==1)  dp[j]=seq1[i-1]==seq2[j-1]?1:dp[j-1];
            else if(j==1)  dp[j]=seq1[i-1]==seq2[j-1]?1:dp[j];
            else{
                if(seq1[i-1]==seq2[j-1]) dp[j]=pre+1;
                else dp[j]=max(dp[j-1],dp[j]);
            }
            pre=temp;
        }
    }
    cout<<dp[n]<<endl;
}
```

Problem 2

0-1 Knapsack Problem

The state transition equation is as below:

$$B[k,w] = \begin{cases} B[k-1,w]+1, & w_k > w \\ max(B[k-1,w], B[k-1,w_k]+b_k), & else. \end{cases} \tag{2}$$

We also adopt method of state compression instead of a traditional approach that takes $O(n^2)$ memory. In details, we use **scrolling array** which is a one-demision array to save each states. For each inside loop, $dp[j]$ represents maxmum value the bag is able to attain if the volume of bag is limited within j respected to subset of S_i. Different from the problem 1, in this question we creat additional array to save states of the last once loop. e.g. $pre[j - weight[i-1]]$ represents maxmum value the bag is able to attain if the volume of bag is limited within j-weight[i-1] respected to subset of S_{i-1}

Hence the optimal method just takes $O(n)$ space complexity and $O(n^2)$ time complexity.

The cpp code is attached as follow:

```cpp
#include<iostream>
#include<vector>
using namespace std;
int main(){
    int n,w;
    cin>>n>>w;
    vector<int> weight,value;
    int w_i,v_i;
    for(int i=0;i<n;i++){
        cin>>w_i>>v_i;
        weight.push_back(w_i);
        value.push_back(v_i);
    }
    vector<int> dp(w+1,0);
    vector<int> pre(w+1,0);
    for(int i=1;i<n+1;i++){
        if(i==n){
            if(weight[i-1]<=w)
            dp[w]=max(dp[w],pre[w-weight[i-1]]+value[i-1]);
            break;
        }
        for(int j=1;j<w+1;j++){
            if(weight[i-1]<=j)
            dp[j]=max(dp[j],pre[j-weight[i-1]]+value[i-1]);
        }
        pre=dp;
    }
    cout<< dp[w] <<endl;
}
```

Problem 3

The Shortest Path Problem

The state transition equation is as below:

$$dst[u] = max\left\{dst[u], dst[v] + w\right\} \tag{3}$$

Since the question descprtion tells that it contains negative edge, we adopt Bellmanford to solve this question. We use two-demision array *edges* to save edge with weight, and one-demision array *dst* to represent the shortest path from start point to each station. Bellmanford claims that the shortest path will not cover than n-1 edges. Therefore in the i-th loop, dst[u] represent the shortest distances to u if only go through i edges.

Hence the optimal method just takes $O(n^2)$ space complexity and $O(n^2)$ time complexity.
The cpp code is attached as follow:

```cpp
#include<iostream>
#include<vector>
#include<cmath>
#include<climits>
using namespace std;
int main(){
    int n,m,s,t;
    cin>>n>>m>>s>>t;
    vector<vector<int>> edges;
    for(int i=0;i<m;i++){
        int u,v,w;
        cin>>u>>v>>w;
        edges.push_back({u,v,w});
    }
    vector<int> dst(n+1,INT_MAX);
    dst[s]=0;
    for(int i=0;i<n-1;i++){
        int flag=0;
        for(const auto& e:edges){
            int u=e[0];
            int v=e[1];
            int w=e[2];
            if(dst[u]!=INT_MAX && dst[v]>dst[u]+w){
                dst[v]=dst[u]+w;
                flag=1;
            }
        }
        if(!flag) break;

    }
    cout<<dst[t]<<endl;
}
```

Problem 4

LIS

Different from what we learn in class, I optimalize the algirithm with less memory and time cost. In this algorithm, we just creat one-demision array to save increasing sequence. In the i-th loop, if $nums[i] > back\ of\ dp$ put the nums[i] into the end of the array. Otherwise find the one which is larger than nums[i] from left, and replace it.

Hence the optimal method just takes $O(n)$ space complexity and $O(nlogn)$ time complexity.
The cpp code is attached as follow:

```cpp
#include<iostream>
#include<vector>
using namespace std;
int main(){
    int n;
    cin>>n;
    vector<int> nums;
    int num;
    for(int i=0;i<n;i++) {
        cin>>num;
        nums.push_back(num);
    }
    if (n <= 1) return n;
    vector<int> dp;
    dp.push_back(nums[0]);
    for (int i = 1; i < n; ++i) {
        if (dp.back() < nums[i]) {
            dp.push_back(nums[i]);
        } else {
            /*     nums[i   ] */
            auto itr = lower_bound(dp.begin(), dp.end(), nums[i]);
            *itr = nums[i];
        }
    }
    cout<<(int)dp.size()<<endl;
}
```

Problem 5

Max M Sum Subsequences Problem

The state transition equation is as below:

$$DP[i,j] = \begin{cases} mk[i-1,j-1] + nums[j], & i == j \\ max(DP[i,j-1] + nums[j], mk[i-1,j-1] + nums[j]), & else. \end{cases} \tag{4}$$

In above equation, $DP[i,j]$ represents max i sum subsequnces of j preflix of sequence when the j-th number is in the i-th subsequence, and $mk[i,j]$ represents max i sum subsequnces of j preflix of the sequence. In order to compress states, we also adopt scrolling arrays to replace above two-demision arrays.

Problem 5 continued on next page...

Hence the optimal method just takes $O(n)$ space complexity and $O(nlogn)$ time complexity.
The cpp code is attached as follow:

```cpp
#include<iostream>
#include<vector>
#include<cmath>
#include<cstring>
using namespace std;
int main(){
    int n,m;
    cin>>n>>m;
    int dp[n];
    int nums[n];
    // vector<int> nums(n,0);
    // vector<int> dp(n,0);
    for(int i=0;i<n;i++){
        int num;
        cin>>num;
        nums[i]=num;
    }
    int res;
    int max_sum;
    // vector<int> mk(n,0);
    int mk[n];
    memset(dp,0,sizeof(int)*n);
    memset(mk,0,sizeof(int)*n);
    for(int i=0;i<m;i++){
        max_sum=INT_MIN;
        for(int j=i;j<n;j++){
            if(j==i) {
                if(j==0) dp[j]=nums[j];
                else dp[j]=mk[j-1]+nums[j];
            }
            else {
                dp[j]=max(dp[j-1]+nums[j],mk[j-1]+nums[j]);
                mk[j-1]=max_sum;
            }
            max_sum=max(max_sum,dp[j]);
        }
    }
    cout<<max_sum<<endl;

}
```