DP基础优化

LIS 最长上升子序列

以某个固定点作为上升子序列的最后一个元素，求出它的最长上升子序列。

取每个固定点的最大值

Dp[i]表示的是：以第i个点作为最后的一个元素的上升子序列的最长长度是多少

```
//O(n^2)做法
for(int i=1;i<=n;i++){
    cin>>a[i];
    dp[i]=1;
    for(int j=1;j<i;j++){
        if(a[i]>a[j]){
            dp[i]=max(dp[i],dp[j]+1);
        }
    }
    ans=max(ans,dp[i]);
}
```

```
//O(nlogn)做法
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+7;
const int inf=0x3f3f3f3f;

int n;
int dp[N],a[N],b[N];

int main(){
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    memset(b,inf,sizeof(b));
    cin>>n;
    int ans=1;
    for(int i=1;i<=n;i++){
```

```
        cin>>a[i];
        int p=lower_bound(b+1,b+ans+1,a[i])-b-1; //找第一个比a[i]大的元素
        dp[i]=p+1; //表示前i位lis长度
        b[dp[i]]=a[i]; //b数组维护一个lis序列
        ans=max(ans,p+1); //更新答案
    }
    return 0;
}
```

最长公共子序列 LCS

Dp[i][j]表示考虑第一个数组的前i位，以及第二个数组的前j位的公共子序列最长长度。

```
//复杂度O(n^2)
#include<bits/stdc++.h>
using namespace std;
const int N=1e3+5;
int n,m;
int dp[N][N];

int main(){
    cin>>n>>m;
    string a,b;
    cin>>a>>b;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            if(a[i-1]==b[j-1]) dp[i][j]=dp[i-1][j-1]+1;
        }else dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
    }
    cout<<dp[n][m]<<endl;
    return 0;
}
```

空间优化

```
for(int i=1;i<=len1;i++){
    pre=0;
    for(int j=1;j<=len2;j++){
        tmp=dp[j];
        if(s1[i-1]==s2[j-1]) dp[j]=pre+1;
        else dp[j]=max(dp[j-1],dp[j]);
        pre=tmp;
    }
}
```

稀疏性优化

最长公共子序列是按位向后比对的，所以a序列每个元素在b序列中的位置如果递增，就说明b中的这个数在a中的这个数整体位置偏后

```
//复杂度O(nlogn)
#include<bits/stdc++.h>
```

```
using namespace std;
const int N=1e3+5;
int n,m;
int dp[N],a[N],b[N],p[N];

int main(){
    cin>>n;
    for(int i=1;i<=n;i++){
        int x;cin>>x;
        p[x]=i;
    }
    for(int i=1;i<=n;i++){
        int x;cin>>x;
        a[i]=p[x];
    }
    int ans=0;
    for(int i=1;i<=n;i++){
        int p=lower_bound(b+1,b+1+ans,a[i])-b-1;
        dp[i]=p+1;
        b[dp[i]]=a[i];
        ans=max(ans,p+1);
    }
    cout<<ans<<"\n";
    return 0;
}
```

bitset优化

手写一个 `bitset`，支持按位与、或、异或、数值相减、位移。

LCS的复杂度降为$O(\frac{nm}{\omega})$

```
/*
 * Author : _Wallace_
 * Source : https://www.cnblogs.com/-Wallace-/
 * Problem : LOJ #6564. 最长公共子序列
 * Standard : GNU C++ 03
 * Optimal : -Ofast
 */
#include <algorithm>
#include <cstddef>
#include <cstdio>
#include <cstring>

typedef unsigned long long ULL;

const int N = 7e4 + 5;
int n, m, u;

struct bitset {
  ULL t[N / 64 + 5];

  bitset() {
    memset(t, 0, sizeof(t));
  }
  bitset(const bitset &rhs) {
```

```
    memcpy(t, rhs.t, sizeof(t));
  }

  bitset& set(int p) {
    t[p >> 6] |= 1llu << (p & 63);
    return *this;
  }
  bitset& shift() {
    ULL last = 0llu;
    for (int i = 0; i < u; i++) {
      ULL cur = t[i] >> 63;
      (t[i] <<= 1) |= last, last = cur;
    }
    return *this;
  }
  int count() {
    int ret = 0;
    for (int i = 0; i < u; i++)
      ret += __builtin_popcountll(t[i]);
    return ret;
  }

  bitset& operator = (const bitset &rhs) {
    memcpy(t, rhs.t, sizeof(t));
    return *this;
  }
  bitset& operator &= (const bitset &rhs) {
    for (int i = 0; i < u; i++) t[i] &= rhs.t[i];
    return *this;
  }
  bitset& operator |= (const bitset &rhs) {
    for (int i = 0; i < u; i++) t[i] |= rhs.t[i];
    return *this;
  }
  bitset& operator ^= (const bitset &rhs) {
    for (int i = 0; i < u; i++) t[i] ^= rhs.t[i];
    return *this;
  }

  friend bitset operator - (const bitset &lhs, const bitset &rhs) {
    ULL last = 0llu; bitset ret;
    for (int i = 0; i < u; i++){
      ULL cur = (lhs.t[i] < rhs.t[i] + last);
      ret.t[i] = lhs.t[i] - rhs.t[i] - last;
      last = cur;
    }
    return ret;
  }
} p[N], f, g;

signed main() {
  scanf("%d%d", &n, &m), u = n / 64 + 1;
  for (int i = 1, c; i <= n; i++)
    scanf("%d", &c), p[c].set(i);
  for (int i = 1, c; i <= m; i++) {
    scanf("%d", &c), (g = f) |= p[c];
    f.shift(), f.set(0);
    ((f = g - f) ^= g) &= g;
```

```
  }
  printf("%d\n", f.count());
  return 0;
}
```

例题

[AHOI2009]CHESS 中国象棋]

求$n\times m$的棋盘上炮互不攻击的方案数。

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
const int mod=9999973;
const int N=205;
int n,m,dp[N][N][N];

signed main(){
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    cin>>n>>m;
    dp[1][m][0]=1;dp[1][m-1][1]=m;dp[1][m-2][2]=m*(m-1)/2; //初始化第一行方案书
    for(int i=2;i<=n;i++){ //枚举行
        for(int j=0;j<=m;j++){ //枚举放0个炮的列数
            for(int k=0;k+j<=m;k++){ //枚举放1个炮的列数
                int l=m-j-k; //放2个炮的列数
                dp[i][j][k]=(dp[i][j][k]+dp[i-1][j][k])%mod; //不放的方案
                if(k>=1&&j+1<=m) dp[i][j][k]=(dp[i][j][k]+dp[i-1][j+1][k-1]*
(j+1))%mod; //放一个炮到空列
                if(l>=1&&k+1<=m) dp[i][j][k]=(dp[i][j][k]+dp[i-1][j][k+1]*
(k+1))%mod; //放一个到有个炮的列
                if(k>=2&&j+2<=m) dp[i][j][k]=(dp[i][j][k]+dp[i-1][j+2][k-2]*
((j+2)*(j+1)/2))%mod;// 放两个到空列。
                if(l>=2&&k+2<=m) dp[i][j][k]=(dp[i][j][k]+dp[i-1][j][k+2]*
((k+2)*(k+1)/2))%mod; //放两个到一个炮的列。
                if(l>=1&&j+1<=m) dp[i][j][k]=(dp[i][j][k]+dp[i-1][j+1][k]*(j+1)*
(k))%mod;
            }
        }
    }
    int ans=0;
    for(int j=0;j<=m;j++){
        for(int k=0;k+j<=m;k++){
            ans=(ans+dp[n][j][k])%mod;
        }
    }
    cout<<ans<<"\n";
    return 0;
}
```

参考资料

https://www.bilibili.com/video/BV11A411A7bQ