ACM-template

langman

December 23, 2017

1 head.set

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<string>
#include<algorithm>
#include<queue>
#include<stack>
#include<vector>
#include<cmath>
#include<set>
#include<cstdlib>
#include<functional>
#include<climits>
#include<cctype>
#include<iomanip>
using namespace std;
typedef long long ll;
#define INF 0x3f3f3f3f
const int mod = 1e9+7 ;
#define clr(a,x) memset(a,x,sizeof(a))
#define cle(a,n) for(int i=1;i<=n;i++) a.clear();
const double eps = 1e-6;
int main()
{
freopen("in.txt","r",stdin);
freopen("out.txt","w",stdout);

return 0;
}
```

2 DP

2.1 LIS/LDS

最大上升子序列,下降,严格上,严格降.
nlogn的复杂度,调用库里面的函数

```
LIS(LDS)
template<class Cmp>
int LIS (Cmp cmp)(nlogn)
{
     static int m, end[N];
    m = 0;
     for (int i=0;i<n;i++)
     {
         int pos = lower_bound(end, end+m, a[i], cmp)-end;
         end[pos] = a[i], m += pos==m;
     }
     return m;
}
     cout << LIS(less<int>()) << endl;          //struct up
     cout << LIS(less_equal<int>()) << endl;    // up
     cout << LIS(greater<int>()) << endl;       //struct dpwn
     cout << LIS(greater_equal<int>()) << endl;//down
```

2.2 dp in bag

背包的题目,是比较基础的dp类型的题
01和完全背包相对来说比较简单,n方 的复杂度
部分背包的话相对来说,因该是把部分背包换成01背包,降低复杂度

他的状态点在于 当背包容量为x时，他的最佳状态 然后找出容量是x的
时候能从 哪几个子状态转移过来。 难点在于：
1：背包的构造
2：背包状态转移方程的寻找
3：方向是从前到后，还是从后到前
4：dp维数的确定

```
// 0 1
for(int i = 0;i<num;i++)
{
   for(int j = v;j>=money[i];j++)
```

```
    {
        dp[j] = max(dp[j],dp[j-money[i]]+value[i]);
    }
}

// part bag (better make it into 0 1 bag)

//full bag
for(int i = 0;i<num;i++)
{
    for(int j = v;j>=money[i];j--)
    {
        dp[j] = max(dp[j],dp[j-money[i]]+value[i]);
    }
}
```