研究生算法课课堂笔记

上课日期：2016 年 12 月 26 日星期一　第(2)节课

组长学号及姓名：1601214502　曹俊杰

组员学号及姓名：1601214518　崔润东

注意：请提交 Word 格式文档

本节课是这学期最后一节算法课，主要讲了考试须知、考试范围及讲解了模拟机试题中股票买卖问题和 Polygon 这两道动态规划题的解题思路。

本课答疑时间：1 月 5 日星期四下午 2-4 点，理科一号楼 1716 会议室，肖老师会答疑到大家没有问题为止。最后祝大家考试顺利，没有挂科

一：考试须知

1：考试必须要带学生证

2：自带笔，考试提供草稿纸

3：缓考申请必须要去学院开具证明

4：严禁作弊，不能替考，不能一台机器登两个账号

二：考试范围

机试一共 3h,十题，占总成绩的 45%

第一题：基础编程题

第二题：难度类似于模拟考中的“红与黑”

第三题：简单分治/简单数据结构

第四题~第六题：中档题，一般来说是课上讲过思路的

第七题~第十题：有一定难度，包括动态规划、图论、分治等方面的内容，难度参考模拟考中的“polygon”

三：股票买卖问题

大致题意：给你 n 天的股票价格，希望买卖两次获得最高的利润。因为允许一天多次买卖，所以问题等价于最多买卖两次而获得最高利润。

问题等价于最大子段和问题。假设股票的价格为$A_1 A_2 \dots A_n$, $B_i = A_{i+1} - A_i$即第 i 天买入第 i+1 天卖出的差价，那么第 i 天买入第 j 天卖出等价于 $A_j - A_i = \sum_{k=i}^{k=j-1} B_k$就是一个子序列求和的问题。

方案 A（肖老师上课讲的）

设$\mathrm{F}(i,k)$表示最多 k 次买卖第 i 天卖出的最大收益，$\mathrm{G}(i,\mathrm{k})$表示最多 k 次买卖前 i 天卖出的最大收益

$$\mathrm{F}(i,k) = max\{G(i,k-1), F(i-1,k) + P_i - P_{i-1}\}$$

$$G(i,k) = max\{G(i-1,k), F(i,k)\}$$

其中$F(i,k)$由$G(i,k-1)$得到的意思是在第 i 天进行一次买卖活动，收益为 0，但是最后一次卖的行为是在第 i 天。

方案 B（UP 主自己的）

设$F(i,k)$表示前 i 天完成了 k 次买卖的最大收益，$G(i,k)$表示前 i 天完成了 k-1 次买卖以及第 k 次的买入操作的最大收益

$$G(i,k) = max\{G(i-1,k), F(i-1,k-1) - P_i\}$$

$$F(i,k) = max\{F(i-1,k), G(i-1,k) + P_i\}$$

自我感觉很好理解...

两个方案本质都是相同的，就看大家更容易理解哪个了

代码都非常短，UP 主就贴自己的代码了：

```
#include <bits/stdc++.h>
using namespace std;
#define INF 0x3fffffff
#define maxn 100005
int a[maxn];
int dp[3][3];
int main(){
    int T;
    cin >> T;
    while (T--){
        int n;
        cin >> n;
        for (int i = 0 ; i <= 2 ; i++)
            for (int j = 0 ; j <= 2 ; j++)
                dp[i][j] = -INF;
        for (int i = 1 ; i <= n ; i++){
            int x;
            scanf("%d",&x);
//顺序很关键，如果从下至上则表示不能
            dp[1][0] = max(dp[1][0],-x);
            dp[1][1] = max(dp[1][1],x + dp[1][0]);
            dp[2][0] = max(dp[2][0],dp[1][1] - x);
            dp[2][1] = max(dp[2][1],x + dp[2][0]);
        }
        cout << dp[2][1] << endl;
    }
        return 0;
}
```

四：Polygon

大致题意：给你一个 n 个点的环，节点上是数字，路径上是+或*运算。现在让你删掉一条边后把环上的点按操作缩成一个点，求能得到的最大值。
思路：类似于求矩阵乘法最小次数，是一个区间 DP，但是有三点不同：

1: 合并代价的计算，矩阵乘法是$\mathrm{F(i,j)} = min\{\, F(i,k) + F(k+1,j) + P_{i-1}P_kP_j \,\}$，而 Polygon 是$F(i,j) = max\begin{cases} F(i,k) + F(k+1,j),\ edge(k,k+1) == + \\ F(i,k) * F(k+1,j),\ edge(k,k+1) == * \end{cases}$

2：考虑到负负得正，如果是乘法的时候有可能两个负的最小值乘起来结果最大

3：如何得到删掉的那一条边，可以将环扩展一倍成一条 2n 的链
假设原图是$A_1A_2 \ldots A_n$，A_iA_{i+1}的边是E_i
新的图是$A'_1A'_2 \ldots A'_{2n}$，$A'_iA'_{i+1}$的边是E'_i
那么$A'_i = A_{(i-1)\,\%\,n+1}, E'_i = E_{(i-1)\,\%\,n+1}$
如果删掉第 i 条边等价于求$A'_{i+1}A'_{i+2} \ldots A'_{i+n}$的最优结果

代码如下：

```
#include <bits/stdc++.h>
using namespace std;
#define maxn 105
int dp[maxn][maxn][2];//[0]表示最大值，[1]表示最小值
char op[maxn];
int num[maxn];
int fmax(int x[],int y[],char z){//计算最大值
    if (z == 't')
        return x[0]+y[0] ;
    else
        return max(x[0]*y[0],x[1]*y[1]);//乘法时考虑负负得正
}
int fmin(int x[],int y[],char z){//计算最小值
    if (z == 't')
        return x[1]+y[1] ;
    else
        return min(x[0]*y[0],min(x[0]*y[1],x[1]*y[0]))//乘法时考虑正负得负
}
int main(){
    int n;
    cin >> n;
    for (int i = 1 ; i <= n ; i++){//将环复制一遍
        cin >> op[i] >> num[i];
```

```
        op[i+n] = op[i];

        num[i+n] = num[i];
    }
    for (int d = 1 ; d <= n ; d++)//枚举长度
        for (int l = 1 ; l <= 2*n-d+1 ; l++){//枚举左界
            int r = l+d-1;//计算得到右界
            if (d == 1)
                dp[l][r][0] = dp[l][r][1] = num[l];//单点时初始化
            else {
            //因为数的范围在[-32768,32767]，所以初始化 max 为下界,min 为上界
                dp[l][r][0] = -32768;
                dp[l][r][1] = 32767;
                for (int k = l + 1 ; k <= r ; k++)//枚举合并点
                    dp[l][r][0] =
max(dp[l][r][0],fmax(dp[l][k-1],dp[k][r],op[k])),
                    dp[l][r][1] =
min(dp[l][r][1],fmin(dp[l][k-1],dp[k][r],op[k]));
            }
    }
    vector<int> ans;
    ans.clear();
    int maxx = -32768;
    for (int l = 1 ; l <= n ; l++)
    if (dp[l][l+n-1][0] > maxx){//如果比当前大则更新
        maxx = dp[l][l+n-1][0];
        ans.clear();
        ans.push_back(l);
    }
    else if (dp[l][l+n-1][0] == maxx){//如果和当前相同则加入
         ans.push_back(l);
    }
    cout << maxx << endl;
    for (int i = 0 ; i < ans.size() ; i++)
        cout << ans[i] << ((i == ans.size()-1) ? '\n' : ' ');
    return 0;
}
```