

8.31 课堂笔记 31

初赛练习

NOIP 提高组 2013 年真题解析

1:

8bit=1Byte (8 个比特位=1 个字节)

8*1024bit=1024Byte(字节)=1KB

1024KB=1MB

1024MB=1GB

1024GB=1TB

b=bit 表示“二进制位/比特位。要么是 0，要么是 1”

B=Byte 表示“字节”

4:

摩尔定律：集成电路上可以容纳的晶体管数目在大约每经过 18 个月便会增加一倍。

图灵：英国人，计算机科学之父，人工智能之父。

姚期智：唯一获得图灵奖的华人，密码学基础，量子计算。

冯·诺依曼：数学家，现代计算机之父，博弈论之父。

本贾尼·斯特劳斯特卢普：c++之父。

欧拉：瑞士数学家、物理学家。贡献：分析力学，柯尼斯堡七桥问题，欧拉公式。

香农：提出了信息熵的概念，为信息论和数字通信奠定了基础。香农奖是通信理论领域最高奖，也被称为“信息领域的诺贝尔奖”。

10:

IPv4：32 位，第一个字节的取值不能为 0，范围是 1-255，后面三个字节可取 0-255

IPv6：128 位

12:

unicode 和 ascii 的区别：

1、ASCII 编码是 1 个字节，而 Unicode 编码通常是 2 个字节。

2、ASCII 是单字节编码，无法用来表示中文；而 Unicode 可以表示所有语言。

3、用 Unicode 编码比 ASCII 编码需要多一倍的存储空间。

多选

2:

类别	排序方法	时间复杂度			空间复杂度	稳定性
		平均情况	最好情况	最坏情况	辅助存储	
插入排序	直接插入	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
	Shell排序	$O(n^{1.3})$	$O(n)$	$O(n^2)$	$O(1)$	不稳定
选择排序	直接选择	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
	堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(1)$	不稳定
交换排序	冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
	快速排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n^2)$	$O(n\log_2 n)$	不稳定
归并排序		$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n)$	稳定
基数排序		$O(d(r+n))$	$O(d(n+rd))$	$O(d(r+n))$	$O(rd+n)$	稳定

注：基数排序的复杂度中， r 代表关键字的基数， d 代表长度， n 代表关键字的个数。

NOIP 提高组 2013 年真题答案

第十九届全国青少年信息学奥林匹克联赛初赛 提高组参考答案

一、单项选择题（共 15 题，每题 1.5 分，共计 22.5 分）

1	2	3	4	5	6	7	8
A	A	B	D	A	B	D	B
9	10	11	12	13	14	15	
D	D	C	B	D	B	B	

二、不定项选择题（共 5 题，每题 1.5 分，共计 7.5 分；每题有一个或多个正确选项，没有部分分）

1	2	3	4	5
AC	AD	CD	AB	ABCD

三、问题求解（共 2 题，每题 5 分，共计 10 分；每题全部答对得 5 分，没有部分分）

- $s_1=0, s_2=1, s_3=1, s_4=1$
- 37/12

四、阅读程序写结果（共 4 题，每题 8 分，共计 32 分）

- Yes
- 133
- 4
- 7

五、完善程序（共计 28 分，以下各程序填空可能还有一些等价的写法，由各省赛区组织本省专家审定及上机验证，可以不上报 CCF NOI 科学委员会复核）

	Pascal 语言	C++语言	C 语言	分值
1.	(1)	$n - p + i$		2
	(2)	$i - p + 1$		2
	(3)	$a[i - p]$		2
	(4)	$j \leq \text{end2}$		3
	(5)	i (或 start2 , 或 $\text{end1} + 1$)		3
	(6)	$j - 1$		3
2.	(1)	$j - 1$		3
	(2)	cur1		3
	(3)	$\text{dec}(\text{count1})$ (或 $\text{count1} := \text{count1} - 1$)	$\text{count1}--$ (或 $\text{count1} = \text{count1} - 1$, 或 $--\text{count1}$)	2
	(4)	$\text{dec}(\text{count2})$ (或 $\text{count2} := \text{count2} - 1$)	$\text{count2}--$ (或 $\text{count2} = \text{count2} - 1$, 或 $--\text{count2}$)	2
	(5)	$\text{cur1} := a[j]$	$\text{cur1} = a[j]$	3

复赛练习

1. P1038

拓扑排序模板

需要留意的地方：

- 只输出大于 0 的输出层（要同时满足这两个条件）
- 输出层的判定可以直接记录出度
- 出度为 0 的一定是输出层
- 在 NULL 的判定中，只判定输出层的 C 是否大于 0
- 如果输出层的 C 都 ≤ 0 ，那么就输出 NULL。
- 如果一个神经元的状态是负的，那么它不会往后传递神经信号

```
#include<bits/stdc++.h>
using namespace std;
int n, p;
int x, y, val;
int numEdge;
int rudu[1000010], chudu[1000010], arr[1000010], u[1000010];

struct edge{
    int w;
    int to;
    int next;
}e[1000010];
int head[1000010];
queue<int> q;

void addEdge(int from, int to, int w){
    numEdge++;
    e[numEdge].next = head[from];
    e[numEdge].to = to;
    e[numEdge].w = w;
    head[from] = numEdge;
}

int main(){
    cin >> n >> p;
    for(int i=1; i<=n; i++){
        cin >> arr[i] >> u[i];
        if(arr[i]>0){
            q.push(i);
        }
    }
    for(int i=1; i<=p; i++){
        cin >> x >> y >> val;
        addEdge(x, y, val);
        rudu[y]++;
        chudu[x]++;
    }
}
```

```

while(!q.empty()){
    int x=q.front();
    q.pop();
    for(int i=head[x]; i!=0; i=e[i].next){
        int y=e[i].to;
        rudu[y]--;
        if(arr[x]>0){
            arr[y] += e[i].w*arr[x];
        }
        if(rudu[y]==0){
            q.push(y);
            arr[y]-=u[y];
        }
    }
}

bool flag=0;
for(int i=1;i<=n;i++){
    if(chudu[i]==0&&arr[i]>0){
        cout<<i<<" "<<arr[i]<<endl;
        flag = true;
    }
}

if(!flag){
    cout<<"NULL";
}

return 0;
}

```

2. P1090

优先队列模板题

priority_queue(), 默认按照从小到大排列。所以 top()返回的是最大值而不是最小值

使用 greater<>后, 数据从大到小排列, top()返回的就是最小值而不是最大值

priority_queue<int, greater<>> pq;//这是错误的

如果使用了第三个参数, 那第二个参数不能省, 用作保存数据的容器

priority_queue<int,vector<int>, greater<>> pq;//这是对的

另一种简单的替代方案是如代码所示, 每次存进去的时候所有数取反, 然后输出结果的时候再次取反

```

#include<bits/stdc++.h>
using namespace std;
int n, num, ans;
int a, b;
priority_queue<int> q;
int main() {
    cin >> n;
    for(int i=1; i<=n; i++) {
        cin >> num;
        q.push(-num);
    }
    while(q.size() >= 2) {
        a = q.top();
        q.pop();
        b = q.top();
        q.pop();
        ans += (-a - b);
        q.push(a + b);
    }
    cout << ans << endl;
    return 0;
}

```

3. P1091

从左往右，按左低右高顺序找出每一个位置左边有几个从低到高的数即为 $b[i]$ （包括自己）

从右往左，按左高右低顺序找出每一个位置右边有几个从高到低的数，即为 $c[i]$ （包括自己）

然后枚举两数之和，求出最小值，值得注意的是从左往右和从右往左都计算了自己，所以计算需要离队的人数时需要减 1，或者输出结果的时候需要 +1

```

//最长上升子序列，最长下降子序列
//顺序上升，逆序上升，对应位置求和，找出最大值，总数-最大值+1即为答案
#include <bits/stdc++.h>
using namespace std;
int a[110], b[110], c[110];
int maxx=1;
int n;
int main() {
    cin >> n;
    for(int i=1; i<=n; i++) {
        cin >> a[i];
    }
    for(int i=1; i<=n; i++) { //顺序，最长上升子序列
        b[i]=1;
        for(int j=1; j<i; j++) {
            if(a[j]<a[i]) {
                if(b[j]+1>b[i]) {
                    b[i]=b[j]+1;
                }
            }
        }
    }
    for(int i=n; i>=1; i--) { //逆序，最长上升子序列
        c[i]=1;
        for(int j=n; j>i; j--) {
            if(a[j]<a[i]) {
                if(c[j]+1>c[i]) {
                    c[i]=c[j]+1;
                }
            }
        }
    }
    for(int i=1; i<=n; i++) {
        if(maxx<b[i]+c[i]) {
            maxx=b[i]+c[i];
        }
    }
    cout << n-maxx+1;
    return 0;
}

```

4. P1052

因为它满足无后效性。

当前的每一个状态都可以由之前的状态转移过来。

当前的决策对之后都不会产生影响。

动态规划最重要的就是状态转移方程，不难得出以下几条结论：

当前的点 i 只与它前 $i-j$ 个点有关

如果当前的点是石头那么就是所有到达该点的位置所需踩的最少石头数加 1

经过上述的分析就可以得出状态转移方程了：

该点为石头：

$$dp[i] = \min(dp[i], dp[i-j] + 1) (s \leq j \leq t)$$

该点不为石头：

$$dp[i] = \min(dp[i], dp[i-j]) (s \leq j \leq t)$$

但本题另一个等待解决的问题是题目的数据范围到了惊人的 $1e9$ ，很难用数组范围把它保存下来，需要使用到离散化的方法。

一个缩小范围是如代码所示的 s ($s+1$)

同时由 NOIP2017 提高组 D1T1 的结论，我们可以知道这个数为 $t=p(p+1)-p-$

$(p+1)t=p(p+1)-p-(p+1)$ 。由于本题的最大步长为 10，因此本题中可以缩小到的极

限距离是 $9 \times 10^9 - 10 = 71$ 。

赛瓦维斯特定理：已知 a, b 为大于 1 的正整数， $\gcd(a, b) = 1$ ，
则使不定方程 $ax + by = C$ 无负数解的最大整数 $C = ab - a - b$

```
/*状态压缩，如果某个数大于s(s+1)，那么它的大小可以缩小到s(s+1)
  缩小的部分可以由s和s+1的加权和表示出来*/
#include<bits/stdc++.h>
using namespace std;
int f[10000010], a[10000010], b[10000010];
bool vis[10000010];
int L, s, t, m;
int ans = 0x3f3f3f3f;
int main() {
    cin >> L;
    cin >> s >> t >> m;
    for (int i = 1; i <= m; i++) {
        cin >> a[i];
    }
    sort(a + 1, a + m + 1);
    if (s == t) { // 特判 s=t 的情况
        int cnt = 0;
        for (int i = 1; i <= m; i++) {
            if (a[i] % s == 0) {
                cnt++;
            }
        }
        cout << cnt;
        return 0;
    }
    for (int i = 1; i <= m; i++) {
        if (a[i] - a[i - 1] > s * (s + 1)) {
            b[i] = b[i - 1] + s * (s + 1); // 状态压缩
        }
        else {
            b[i] = b[i - 1] + a[i] - a[i - 1];
        }
        vis[b[i]] = true;
    }
    memset(f, 0x3f, sizeof f);
    f[0] = 0;
    L = b[m] + 10; // 最后一步跳的距离不肯能超过10
    for (int i = s; i <= L; i++) {
        for (int j = s; j <= min(t, i); j++) {
            f[i] = min(f[i], f[i - j] + vis[i]);
        }
    }
    for (int i = b[m]; i <= L; i++) {
        ans = min(ans, f[i]);
    }
    cout << ans;
    return 0;
}
```

5. P1064

01 背包的决策是

- 不选，然后去考虑下一个
- 选，背包容量减掉那个重量，总值加上那个价值。

这个题的决策是五个，分别是：

- 不选，然后去考虑下一个
- 选且只选这个主件
- 选这个主件，并且选附件 1
- 选这个主件，并且选附件 2
- 选这个主件，并且选附件 1 和附件 2。


```

#include <iostream>
#define maxn 32005
using namespace std;
int n, m;
int v, p, q;
int main_item_w[maxn]; //主件费用
int main_item_c[maxn]; //主件价值
int annex_item_w[maxn][3]; //附件费用
int annex_item_c[maxn][3]; //附件价值
int f[maxn];
int main() {
    cin >> n >> m;
    for (int i=1; i<=m; i++) {
        cin >> v >> p >> q;
        //主件
        if (!q) {
            main_item_w[i] = v;
            main_item_c[i] = v * p; //保存价值: 价格*重要度
        }
        /*附件:
        [q][ ]: q表示依附于哪个主件
        [q][0]: 第几个附件 1或2
        w[q][[q][0]]: 价格
        c[q][[q][0]]: 价值 (价格*重要度)
        */
        else {
            annex_item_w[q][0]++; //统计该物品附件的个数
            annex_item_w[q][annex_item_w[q][0]] = v;
            annex_item_c[q][annex_item_w[q][0]] = v * p;
        }
    }

    for (int i=1; i<=m; i++)
        for (int j=n; main_item_w[i]!=0 && j>=main_item_w[i]; j--) { //必须为主件
            f[j] = max(f[j], f[j-main_item_w[i]]+main_item_c[i]);

            if (j >= main_item_w[i] + annex_item_w[i][1])
                f[j] = max(f[j], f[j-main_item_w[i]-annex_item_w[i][1]]+main_item_c[i]+annex_item_c[i][1]);

            if (j >= main_item_w[i] + annex_item_w[i][2])
                f[j] = max(f[j], f[j-main_item_w[i]-annex_item_w[i][2]]+main_item_c[i]+annex_item_c[i][2]);

            if (j >= main_item_w[i] + annex_item_w[i][1] + annex_item_w[i][2])
                f[j] = max(f[j], f[j-main_item_w[i]-annex_item_w[i][1]-annex_item_w[i][2]]+main_item_c[i]+annex_item_c[i][1]+annex_item_c[i][2]);
        }

    cout << f[n] << endl;
    return 0;
}

```

6. P1063

状态转移方程的推导如下

一、将珠子划分为两个珠子一个区间时，这个区间的能量=左边珠子*右边珠子*右边珠子的下一个珠子

二、区间包含 3 个珠子，可以是左边单个珠子的区间+右边两珠子的区间，或者左边两珠子的区间右边+单个珠子的区间

即，先合并两个珠子的区间，释放能量，加上单个珠子区间的能量（单个珠子没有能量）

Energy=max(两个珠子的区间的能量+单个珠子区间的能量，单个珠子的区间的能量+两个珠子的区间的能量)

三、继续推 4 个珠子的区间，5 个珠子的区间。

于是可以得到方程：Energy=max（不操作的能量，左区间合并后的能量+右区间合并后的能量+两区间合并产生能量）

两区间合并后产生的能量=左区间第一个珠子*右区间第一个珠子*总区间后面的一个珠子

```

#include <iostream>
using namespace std;
int _n = 0;
int _num[210];
int _sum[210];
int _dp[210][210];
int _maxi = 0;
int main () {
    cin >> _n;
    for(int i = 1; i <= _n; i++) {
        cin >> _num[i];
        _num[i + _n] = _num[i]; //环转链
    }
    int j = 0;
    for(int len = 3; len <= _n + 1; len++) { //枚举区间长度 len=3时, 表示两颗珠子的聚合
        for(int i = 1; i + len - 1 <= 2 * _n; i++) { //枚举区间起点
            j = i + len - 1; //区间终点
            for(int k = i + 1; k <= j - 1; k++) { //枚举分割点 因为枚举的是珠子之间的聚合, 所以不能等于i和j
                _dp[i][j] = max(_dp[i][j], _dp[i][k] + _dp[k][j] + _num[i] * _num[k] * _num[j]);
            }
        }
    }
    //环形比较
    for(int i = 1; i <= _n; i++) {
        _maxi = max(_maxi, _dp[i][i + _n]); //长度为n+1
    }
    cout << _maxi;
    return 0;
}

```