Daily Notes

Normal Notes

acm International Collegiate Programming Contest

Lee

2021.11.01

目　录

1 章节 1

宋体 模板[1]

脚注模板[2]。

```
#include<stdio.h>
#include<iostream>
// A comment xxxxxx xxxxxx xxxxxx xxxxxx xxxxxx xxxxxx
/*
xxx
xxx
xxxxxx
xx
xxxx
xxxxxxxxxxx
xxxxxxxxx
xxxx
*/
int main(void)
{
printf("Hello World\n");
return 0;
}
```

黑体

仿宋

楷书

[1] **收稿日期**：2000-06-30；

[2] **收稿日期**：2000-06-30；**修回日期**：2000-11-16

基金项目：“九五” 国家科技攻关资助项目 (96-B02-03-05)

作者简介：XXX(1970-)，男，中国科学院资源与环境信息系统国家重点实验室博士后，主要从事交通网络的地理信息系统数据模型和网络分析相关算法研究。

2 基础算法

2.1 排序

2.1.1 快速排序

```
void quick_sort(int q[], int l, int r) {
    if (l >= r) return;
    int i = l - 1, j = r + 1, x = q[ l + r >> 1 ];
    while (l < r) {
        do i++; while (q[i] < x);
        do j--; while (q[j] > x);
        if (i < j) swap(q[i], q[j]);
    }
    quick_sort(q, l, j);
    quick_sort(q, j + 1, r);
}
```

2.1.2 归并排序

```
void merge_sort(int q[], int l, int r) {
    if (l >= r) return;
    int mid = l + r >> 1;
    merge_sort(q, l, mid);
    merge_sort(q, mid + 1, r);
    int i = l, j = mid + 1, k = 0;
    while (i <= mid && j <= r) {
        if (q[i] < q[j]) tmp[k++] = q[i++];
        else tmp[k++] = q[j++];
    }
    while (i <= mid) tmp[k++] = q[i++];
    while (j <= r) tmp[k++] = q[j++];
    for (int i = l, j = 0; i <= r; i++, j++) q[i] = tmp[j];
}
```

2.2 字符串

2.2.1 KMP

```
for (int i = 1, j = 0; i <= plen; i++) {
    j = ne[j];
    // i表示str串的位置
    // j表示pattern串匹配的位置
    while (j && p[i] != p[j]) j = ne[j];
    ne[i+1] = (p[i] == p[j]) ? j + 1 : 0;
}

for (int i = 0, j = 0; i < slen; i++) {
    while (j && s[i] != p[j]) j = ne[j];
    if (s[i] == p[j]) j++;
    if (j == plen) {
        j = ne[j];
        // match
    }
}
```

2.2.2 Manacher

3 数据结构

4 搜索与图论

5 数论

5.1 质数

5.1.1 埃式筛

从 2 开始筛掉质数本身的倍数，向后找到的第一个没被筛掉的数一定是质数，复杂度 $O(nloglogn)$

```
void get_primes(int n) {
    for (int i = 2; i <= n; i++) {
        if (!st[i]) {
```

```
            primes[cnt++] = i;
            for (int j = i + i; j <= n; j += i) st[j] = true;
        }
    }
}
```

5.1.2 线性筛

对于质数 i，从当前已经找到的第一个质数开始筛 $primes[j] * i$，直到 $primes[j] \mid i$，复杂度 $O(n)$

```
void get_primes(int n) {
    for (int i = 2; i <= n; i++) {
        if (!st[i]) {
            primes[cnt++] = i;
        }
        for (int j = 0; primes[j] <= n / i; j++) {
            st[primes[j] * i] = true;
            if (i % primes[j] == 0) break;
        }
    }
}
```

6 动态规划

6.1 背包问题

6.1.1 分组背包问题

问题描述：多组物品，每组有若干个物品，同组内的物品最多只能选一个，在总体积不超过限制的情况下，求总价值。

- 状态表示 $f[i, j]$
  - 集合：从前 i 组物品中选，且总体积不大于 j 的所有选法
  - 属性：所有选法价值最大值（Max）
- 状态计算 – 集合划分：按第 i 组物品选哪个作为划分依据
  1. 第 i 组物品一个都不选：$f[i-1, j]$

2. 第 i 组物品选择第 1 件物品：$f[i-1, j-v[i,1]]+w[i,1]$
3. ...
4. 第 i 组物品选择第 k 件物品：$f[i-1, j-v[i,k]]+w[i,k]$
5. ...

由此可推出分组背包的状态转移方程为：

$$f[i][j] = max(f[i-1][j-v[i,k]]+w[i,k])$$

一维优化：从大到小枚举空间 j，由于只申请了一维空间存储最大价值，但是每一轮计算需要使用上一轮 $i-1$ 的旧值进行更新，所以 j 需要反方向遍历，$j := m \to 0$ && $j \geq v[i,k]$

$$f[j] = max(f[j-v[i,k]]+w[i,k])$$

```
for i ← 1 to n do
    for j ← m to 0 do
        for k ← 0 to s[i] do
            if j ≥ v[i][k] then
                f[i] = max(f[j], f[j − v[i][k]] + w[i][k]);
            end
        end
    end
end
```

题目. AcWing 9. 有 N 个物品和一个容量是 V 的背包。每件物品有若干个，同一组的物品最多只能选一个。每件物品的体积是 v_{ij}，价值是 w_{ij}，其中 i 是组号，j 是组内编号。求将哪些物品装入背包，可使物品总体积不超过背包容量，且总价值最大。

代码模板.

```cpp
#include <iostream>
using namespace std;
const int N = 105;
```

```
4  int f[N];
5  int v[N][N], w[N][N], s[N];
6  int n, m;
7  int main() {
8      cin >> n >> m;
9    // 读入数据
10     for (int i = 1; i <= n; i++) {
11         cin >> s[i];
12         for (int j = 1; j <= s[i]; j++) {
13             cin >> v[i][j] >> w[i][j];
14         }
15     }
16     for (int i = 1; i <= n; i++) {
17         for (int j = m; j >= 0; j--) {
18             for (int k = 0; k <= s[i]; k++) {
19                 if (j >= v[i][k]) f[j] = max(f[j], f[j - v[i][k]] + w[i][k
                       ]);
20             }
21         }
22     }
23     cout << f[m] << endl;
24     return 0;
25 }
```

6.2 线性 DP

6.2.1 数字三角形

题目. AcWing 898. 给定一个如下图所示的数字三角形，从顶部出发，在每一结点可以选择移动至其左下方的结点或移动至其右下方的结点，一直走到底层，要求找出一条路径，使路径上的数字的和最大。

```
1        7
2      3   8
```

```
    8   1   0
  2   7   4   4
4   5   2   6   5
```

分析.

- 状态表示 $f[i,j]$
  - 集合：所有从起点，走到 (i,j) 的路径集合
  - 属性：所有路径上数字之和的最大值 Max
- 状态计算 – 集合划分：按其前驱路径方向划分
  1. 来自左上：$f[i-1,j-1]+a[i,j]$
  2. 来自右上：$f[i-1,j]+a[i,j]$

  由此可推出状态转移方程为：

$$f[i][j]=max(f[i-1][j-1]+a[i][j],f[i-1][j]+a[i][j])$$

代码模板.

```cpp
#include <iostream>
using namespace std;
const int N = 505, INF = 1e9;
int a[N][N], f[N][N];
int n;
int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= i; j++) {
            scanf("%d", &a[i][j]);
        }
    }
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= i + 1; j++) {
```

```
15              f[i][j] = -INF;
16          }
17      }
18      f[1][1] = a[1][1];
19      for (int i = 2; i <= n; i++) {
20          for (int j = 1; j <= i; j++) {
21              f[i][j] = max(f[i-1][j-1] + a[i][j], f[i-1][j] + a[i][j]);
22          }
23      }
24      int res = -INF;
25      for (int i = 1; i <= n; i++) res = max(res, f[n][i]);
26      printf("%d\n", res);
27      return 0;
28  }
```

6.2.2 最长上升子序列

题目. AcWing 895. 给定一个长度为 N 的数列，求数值严格单调递增的子序列的长度最长是多少。

分析.

- 状态表示 $f[i]$
  - 集合：所有以第 i 个数结尾的上升子序列集合
  - 属性：所有以 i 结尾的上升子序列的长度最大值 Max
- 状态计算 – 集合划分：以子序列的倒数第二个数的位置作为划分依据
  1. 0 表示序列长度为 1，即没有倒数第二个数：0
  2. 1 表示倒数第二个数是 $a[1]$：$f[1]+1$
  3. ...
  4. j 表示倒数第二个数是 $a[j]$：$f[j]+1$
  5. ...

  由此可推出状态转移方程为：

$$f[i] = max(f[j] + 1) \ ; \ j = 0, 1, 2, ..., i - 1 \ \&\& \ a[j] < a[i]$$

代码模板.

```
#include <iostream>
using namespace std;
const int N = 1005;
int a[N], f[N];
int n;
int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
    for (int i = 1; i <= n; i++) {
        f[i] = 1;
        for (int j = 1; j < i; j++) {
            if (a[j] < a[i]) f[i] = max(f[i], f[j] + 1);
        }
    }
    // 最大值不一定是以最后一个元素结尾
    int res = 0;
    for (int i = 1; i <= n; i++) res = max(res, f[i]);
    printf("%d\n", res);
    return 0;
}
```

6.2.3 最长上升子序列 II

题目. AcWing 896. 给定一个长度为 N 的数列，求数值严格单调递增的子序列的长度最长是多少。

分析.

贪心、二分

上升序列按长度分类，各个长度存结尾最小的数，即每种长度的最长上升子序列结尾的最小值；结尾数值严格单调递增。

各个长度的结尾最小数值构成一个递增队列。

对于一个数 a_i 应该接到小于它的最大一个数后面（二分查找），可能有两种情况：

1. 更新某个值：不改变最长子序列的长度，但是会更新某个长度下的最长上升子序列的结尾最小值
2. 插入（追加）到队列最后：获得一个新长度的上升子序列和它的结尾最小值

代码模板.

```cpp
#include <iostream>
using namespace std;
const int N = 100005;
int a[N], q[N];
int n;
int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
    int len = 0;
    q[0] = -1e9; // 边界条件
    for (int i = 1; i <= n; i++) {
        int l = 0, r = len;
        while (l < r) {
            int mid = l + r + 1 >> 1;
            if (q[mid] < a[i]) l = mid;
            else r = mid - 1;
        }
        len = max(len, r + 1);
        q[r + 1] = a[i];
    }
    // 小于a[i]的最后一个数
    printf("%d\n", len);
    return 0;
}
```

7 日常杂项

7.1 数学方法

7.1.1 四舍五入与取整

- 四舍五入：$p/k+0.5$
- 向下取整：p/k （默认向下取整）
- 向上取整：$(p+k-1)/k$

向上取整证明：假设 $p=k*n+r$，$(p,k,n,r)\in\mathbb{Z}\quad\&\quad 0\le r\le k-1)$，对 k 能否整除 p 分两种情况讨论：

1. 如果 $k\mid p$，即 $r=0$，$p=k*n$，则 p/k 向上取整结果应该是 n；
   此时有：$p+k-1=k*n+k-1$

$$\begin{aligned}\frac{p+k-1}{k}&=\frac{k*n+k-1}{k}\\&=n+1-\frac{1}{k}\end{aligned}$$

$$n<\frac{p+k-1}{k}<n+1$$

   向下取整计算结果是 n，符合条件。
2. 如果 $k\nmid p$，即 $r\neq 0$，$p=k*n+r$，则 p/k 向上取整的结果应该是 $n+1$；
   此时有：$p+k-1=k*n+r+k-1$
   因为：$1\le r<k$，所以 $k\le r+k-1<2k-1$，有 $1\le\frac{r+k-1}{k}<2-\frac{1}{k}$

$$\begin{aligned}\frac{p+k-1}{k}&=\frac{k*n+r+k-1}{k}\\&=n+\frac{r+k-1}{k}\end{aligned}$$

$$n+1\le\frac{p+k-1}{k}<n+2$$

   向下取整计算结果是 $n+1$，符合条件。

A 附录 1

A.1 附录 1-1