2025暑期集训第4场

T1.加分二叉树

- 区间DP
- 定义$f[l,r]$表示$s[l:r]$构成的二叉树的最高得分。
- 在$[l:r]$枚举子树的根$x，显然有

$$f(l,r)=\max_{l\le x\le r} f(l,x-1)\times f(x+1,r)$$

  注意：为保证合法，当$l>r$时，有$f(l,r)=1$，表示空子树。
- 如何得到二叉树的前序遍历？在状态转移时记录$f[l][r]$的根。

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N = 35;
int d[N];
// f[l][r]表示区间[l,r]建树的最高分
// rt[l][r]表示区间[l,r]最高分时以谁为根
int f[N][N], rt[N][N];

void dfs(int l, int r)
{
    if(rt[l][r] == 0) return;
    cout << rt[l][r] << " ";
    dfs(l, rt[l][r] - 1), dfs(rt[l][r] + 1, r);
}

int main()
{
    int n; cin >> n;
    for(int i = 1; i <= n; ++i) cin >> d[i];
    memset(rt, 0, sizeof(rt));
    for(int i = 0; i <= n + 1; ++i)
        for(int j = 0; j <= n + 1; ++j)
            f[i][j] = 1;
    for(int i = 1; i <= n; ++i) f[i][i] = d[i], rt[i][i] = i;
    for(int len = 2; len <= n; ++len)
        for(int l = 1; l <= n - len + 1; ++l)
            for(int r = l + len - 1, k = l; k <= r; ++k)
            {
                int t = f[l][k - 1] * f[k + 1][r] + d[k];
                if(f[l][r] < t) f[l][r] = t, rt[l][r] = k;
            }
    cout << f[1][n] << "\n";
    dfs(1, n);
    return 0;
}
```

T2.环路运输

- 破环为链，将原来n座仓库的信息复制一份接到原序列后，形成长度为$2n$的序列。
- 为了处理方便，不妨令$i > j$，那么$i - j \leq \frac{n}{2}$，因为如果$i - j > \frac{n}{2}$，显然从原始环路的另一侧更近，此时令$j = j + n$，相当于$j - i \leq \frac{n}{2}$。
- 于是问题转化为：找到合适的i, j，满足$i - j \leq \frac{n}{2}$且$a[i] + a[j] + i - j$的值最大。
- 从小到大枚举i，显然只需要找到$i - j \leq \frac{n}{2}$且$a[j] - j$最大的值即可，显然可以用单调队列。
- 时间复杂度：O(N)。

```cpp
#include <bits/stdc++.h>
using namespace std;

const int N = 2e6 + 10;
int a[N], q[N];

int main()
{
    int n; scanf("%d", &n);
    for(int i = 1; i <= n; ++i)
        scanf("%d", &a[i]), a[i + n] = a[i];
    int l = 1, r = 1, ans = 0;
    q[1] = 1;
    for(int i = 2; i <= 2 * n; ++i)
    {
        while(l <= r && i - q[l] > n / 2) ++l;
        ans = max(ans, a[i] + i + a[q[l]] - q[l]);
        while(l <= r && a[i] - i >= a[q[r]] - q[r]) --r;
        q[++r] = i;
    }
    printf("%d\n", ans);
    return 0;
}
```

T3.战略游戏

- 树形DP
- 每个结点可以放置或者不放置士兵，定义$f(x,0)$表示以x根的子树在x不放置士兵的情况下的最小值，定义$f[x][1]$表示以x根的子树在x放置士兵的情况下的最小值。
- 如果不在x放置士兵，那么x的孩子必须放置士兵，显然有

$$f(x,0)=\sum_{y\in Son(x)}f(y,1)$$

- 如果在x放置士兵，那么x的孩子可以放或者不放士兵，显然有

$$f(x,1)=\sum_{y\in Son(x)}\{f(y,0),f(y,1)\}$$

- 时间复杂度：O(N)。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1510;
vector<int> g[N];
int n;
// f[x][0]表示以x根的子树在x不放置士兵的情况下的最小值
// f[x][1]表示以x根的子树在x放置士兵的情况下的最小值
int f[N][2];

void init()
{
    for(int i = 0; i <= n - 1; ++i) g[i].clear();
    for(int i = 0; i <= n - 1; ++i)
    {
        int x, s; scanf("%d:(%d)", &x, &s);
        if(!s) continue;
        for(int j = 1; j <= s; ++j)
        {
            int y; scanf("%d", &y);
            g[x].push_back(y), g[y].push_back(x);
        }
    }
}

void dp(int x, int fa)
{
    f[x][0] = 0, f[x][1] = 1;
    for(auto y: g[x])
    {
        if(y == fa) continue;
        dp(y, x);
        f[x][0] += f[y][1];
        f[x][1] += min(f[y][0], f[y][1]);
    }
}

int main()
```

```
{
    while(scanf("%d", &n) != EOF)
    {
        init();
        memset(f, 0, sizeof(f));
        dp(0, -1);
        printf("%d\n", min(f[0][0], f[0][1]));
    }
    return 0;
}
```

T4.方块消除

- 区间DP
- 定义$f(l,r)$表示$[l:r]$消除能产生的最高分，但是方块消除后后续方块会左移，无法描述。
- 定义$f(l,r,k)$表示$[l:r]$后有k个和r相同的块消除后能产生的最高分。
- 注意：将原始的方块相同的合并在一起(总共m个区域)，定义$b[i]$表示颜色，$c[i]$表示区域大小。
- 如果$b[r]$选择和后续的k个块合并，显然有：

$$f(l,r,k)=\max\{f(l,r,k),f(l,r-1,0)+(c[r]+k)^2\}$$

- 如果选择$[l:r]$内的一块合并，使得$[l:r]$内的和$b[r]$相同的块与$b[r]$相连，那么有：

$$f(l,r,k)=\max_{l\le x<r,b[x]=b[r]}\{f(l,r,k),f(l,x,k+c[r])+f(x+1,r-1,0)\}$$

- 初始状态：$f(i,i,k)=(c[i]+k)^2$。

```cpp
#include <bits/stdc++.h>
using namespace std;

const int N = 210;
int a[N], b[N], c[N];
long long f[N][N][N];

int main()
{
    int n; scanf("%d", &n);
    for(int i = 1; i <= n; ++i) scanf("%d", &a[i]);
    int m = 0; // 合并之后剩余的块数
    for(int i = 1; i <= n; ++i)
        if(a[i] != a[i - 1]) b[++m] = a[i], c[m] = 1;
        else ++c[m];
    for(int i = 1; i <= m; ++i)
        for(int k = 0; k <= n; ++k)
            f[i][i][k] = (c[i] + k) * (c[i] + k);
    for(int len = 2; len <= m; ++len)
        for(int l = 1; l <= m - len + 1; ++l)
            for(int r = l + len - 1, k = 0; k <= n; ++k)
            {
                f[l][r][k] = f[l][r - 1][0] + (c[r] + k) * (c[r] + k);
                for(int x = l; x < r; ++x)
                    if(b[x] == b[r])
                        f[l][r][k] = max(f[l][r][k], f[l][x][c[r] + k] + f[x + 1][r - 1][0]);
            }
    printf("%lld", f[1][m][0]);
    return 0;
}
```

T5.外卖

- 背包类树形DP
- 定义$f(x,i,0/1)$表示在x为根的子树中花费i时间不回到/回到x能取得的最大价值。
- 显然有$f(x,0,0/1)=0,f(x,1,0/1)=a[x]$。
- 如果回到x，那么对于子树如果到子树y花费了j步，整个流程相当于：$x\to y\to x\to$ 其他子树 $\to x$。

$$f(x,i,1)=\max\{f(y,j-2,1)+f(x,i-j,1)\}$$

- 如果不会到x，那么如果最终停在y，整个流程相当于：$x\to$ 其他子树 $\to x\to y$。

$$f(x,i,0)=\max\{f(x,i-j,1)+f(y,j-1,0)\}$$

- 如果不回到x，那么最终停在其他子树中，整个流程相当于：$x\to y\to x\to$ 其他子树。

$$f(x,i,0)=\max\{f(x,i-j,0)+f(y,j-2,1)\}$$

```cpp
#include <bits/stdc++.h>
using namespace std;

const int N = 510;
int f[N][2 * N][2];
int a[N];
int head[N], ver[2 * N], nxt[2 * N];
int tot = 0;
int n, k;

void add(int x, int y)
{
    ver[++tot] = y;
    nxt[tot] = head[x], head[x] = tot;
}

void dp(int x, int fa)
{
    f[x][1][0] = f[x][1][1] = a[x];
    for(int z = head[x]; z; z = nxt[z])
    {
        int y = ver[z];
        if(y == fa) continue;
        dp(y, x);
        for(int i = k; i >= 1; --i)
            for(int j = 1; j <= i; ++j)
            {
                f[x][i][0] = max(f[x][i][0], f[x][i - j][1] + f[y][j - 1][0]); // x->其他->x->y
                if(j >= 2)
                {
                    f[x][i][0] = max(f[x][i][0], f[y][j - 2][1] + f[x][i - j][0]); // x->y->x->其他
                    f[x][i][1] = max(f[x][i][1], f[y][j - 2][1] + f[x][i - j][1]); // x->y->x->其他->x
                }
            }
    }
}
```

```
int main()
{
    while(scanf("%d%d", &n, &k) != EOF)
    {
        memset(a, 0, sizeof(a));
        memset(f, 0, sizeof(f));
        memset(head, 0, sizeof(head));
        memset(ver, 0, sizeof(ver));
        memset(nxt, 0, sizeof(nxt));
        tot = 0;
        for(int i = 1; i <= n; ++i) scanf("%d", &a[i]);
        for(int i = 1; i <= n - 1; ++i)
        {
            int x, y; scanf("%d%d", &x, &y);
            add(x, y), add(y, x);
        }
        dp(1, 0);
        printf("%d\n", max(f[1][k][0], f[1][k][1]));
    }
    return 0;
}
```