动态规划与搜索

目录

单调队列优化的斜率DP

```
int N, M;
long long sum[500010], dp[500010];
struct Queue {
    long long x;
    long long y;
};
Queue queue[500010];
inline int nextInt() {
    char c;
    c = getchar();
    while (c != '-' && (c < '0' || c > '9'))
        c = getchar();
    int n = 0, s = 1;
    if (c == '-')
        s = -1, c = getchar();
    while (c >= '0' && c <= '9')
        n *= 10, n += c - '0', c = getchar();
    return n * s;
}
inline long long cross(Queue a, Queue b, Queue c) // ab x bc
{
    Queue p, q;
    p.x = b.x - a.x;
    p.y = b.y - a.y;
    q.x = c.x - b.x;
    q.y = c.y - b.y;
    return p.x * q.y - q.x * p.y;
}
inline int isLeft(Queue a, Queue b, Queue c) {
    return cross(a, b, c) > 0;//"="不能有!!!!!
}
inline int larger(Queue a, Queue b, long long k) // a->c  >  b->c
{
    return a.y - k * a.x > b.y - k * b.x;
}
void init() {
    sum[0] = 0;
    for (int i = 1; i <= N; i++) {
        int t = nextInt();
        sum[i] = sum[i - 1] + t;
    }
}
void work() {
    int head = 0, rear = 0;
    dp[0] = 0;
    dp[1] = sum[1] * sum[1] + M;
    for (int i = 2; i <= N; i++) {
        Queue current;
        current.x = sum[i - 1];
        current.y = dp[i - 1] + sum[i - 1] * sum[i - 1];
        for (int j = rear - 1; j >= head; j--) {
            if (rear - head >= 2 && !isLeft(queue[j - 1], queue[j],
current))
                rear--;
            else
                break;
        }
        queue[rear++] = current;
```

```
        while (rear - head >= 2 && !larger(queue[head + 1],
queue[head], 2
                * sum[i]))
            head++;
        dp[i] = queue[head].y - 2 * sum[i] * queue[head].x + M +
sum[i]
                * sum[i];
        dp[i] = min(dp[i], sum[i] * sum[i] + M);
    }
    printf("%I64d\n", dp[N]);
}
```

单调队列的2D1D斜率DP

```
int N, M;
long long s[1010], t[1010], dp[1010][1010];
struct Queue {
    long long x, y;
};
Queue queue[1010];
inline int nextInt() {
    char c;
    c = getchar();
    while (c != '-' && (c < '0' || c > '9'))
        c = getchar();
    int n = 0, s = 1;
    if (c == '-')
        s = -1, c = getchar();
    while (c >= '0' && c <= '9')
        n *= 10, n += c - '0', c = getchar();
    return n * s;
}
inline int isLeft(Queue a, Queue b, Queue c) {
    long long px = b.x - a.x;
    long long py = b.y - a.y;
    long long qx = c.x - b.x;
    long long qy = c.y - b.y;
    return px * qy > qx * py;
}
inline int larger(Queue a, Queue b, int k) {
    return a.y - k * a.x > b.y - k * b.x;
}
void init() {
    for (int i = 1; i <= N; i++) {
        int m = nextInt();
        s[i] = s[i - 1] + m;
        t[i] = t[i - 1] + m * m;
    }
}
void work() {
    for (int i = 1; i <= N; i++)
        dp[0][i] = (s[i] * s[i] - t[i]) / 2;
    for (int i = 1; i <= M; i++) {
        int head = 0, rear = 0;
        dp[i][i] = 0;
        for (int j = i + 1; j <= N; j++) {
            Queue current;
            current.x = s[j - 1];
            current.y = (s[j - 1] * s[j - 1] + t[j - 1]) / 2 + dp[i -
1][j - 1];
            for (int k = rear - 1; k >= head; k--) {
                if (rear - head >= 2
                        && !isLeft(queue[k - 1], queue[k], current))
```

```
                rear--;
            else
                break;
        }
        queue[rear++] = current;
        while (rear - head >= 2 && larger(queue[head], queue[head
+ 1],
                s[j]))
            head++;
        dp[i][j] = queue[head].y - s[j] * queue[head].x + (s[j] *
s[j]
                - t[j]) / 2;
        dp[i][j] = min(dp[i][j], dp[i - 1][i - 1] + (((s[j] - s[i
- 1])
                * (s[j] - s[i - 1]) - t[j] + t[i - 1]) / 2));
    }
  }
  long long res = 0x7FFFFFFFFFFFFFFFLL;
  for (int i = 0; i <= M; i++)
    res = min(res, dp[i][N]);
  printf("%I64d\n", res);
}
```

查找树优化斜率DP

```
#define sig(t) (fabs((t))<1e-9?0:((t)>0?1:-1))

#define maxn 100010
inline double nextDouble() {
    char c;
    c = getchar();
    while (c != '-' && c != '.' && (c < '0' || c > '9'))
        c = getchar();
    int n = 0, s = 1;
    if (c == '-')
        s = -1, c = getchar();
    while (c >= '0' && c <= '9')
        n *= 10, n += c - '0', c = getchar();
    double m = (double) n, t = 1.0;
    if (c == '.')
        c = getchar();
    else
        return m * s;
    while (c >= '0' && c <= '9')
        t /= 10, m += t * (c - '0'), c = getchar();
    return m * s;
}
struct Queue {
    double x;
    double y;

    double left, right;

    bool operator<(Queue &p) {
        return sig(x-p.x) < 0;
    }

    bool operator==(Queue &p) {
        if (!sig(x-p.x))
            return 1;
        return 0;
    }
};
```

```
template<class T>
struct Treap {
    int C, Top, bin[maxn];
    int L[maxn], R[maxn], P[maxn], pri[maxn], size[maxn];
    T key[maxn];

    int root;

    Treap() {
        srand(time(0));
        root = 0;
        C = 0;
        Top = 0;
        memset(L, 0, sizeof(L));
        memset(R, 0, sizeof(R));
        memset(P, 0, sizeof(P));
        memset(key, 0, sizeof(key));
        memset(size, 0, sizeof(size));
    }
    void rightRotate(int x) {
        int y = L[x];
        L[x] = R[y];
        if (R[y])
            P[R[y]] = x;
        P[y] = P[x];
        if (!P[x])
            root = y;
        else if (x == L[P[x]])
            L[P[x]] = y;
        else
            R[P[x]] = y;
        R[y] = x;
        P[x] = y;
    }
    void leftRotate(int x) {
        int y = R[x];
        R[x] = L[y];
        if (L[y])
            P[L[y]] = x;
        P[y] = P[x];
        if (!P[x])
            root = y;
        else if (x == L[P[x]])
            L[P[x]] = y;
        else
            R[P[x]] = y;
        L[y] = x;
        P[x] = y;
    }
    int min(int x) {
        while (L[x])
            x = L[x];
        return x;
    }
    int max(int x) {
        while (R[x])
            x = R[x];
        return x;
    }
    int next(int x) {
        if (R[x])
            return min(R[x]);
        int y = P[x];
```

```
    while (y && x == R[y]) {
        x = y;
        y = P[y];
    }
    return y;
}
int pre(int x) {
    if (L[x])
        return max(L[x]);
    int y = P[x];
    while (y && x == L[y]) {
        x = y;
        y = P[y];
    }
    return y;
}
void makeSize(int x) {
    if (!L[x] && !R[x])
        size[x] = 1;
    else {
        if (L[x])
            makeSize(L[x]);
        if (R[x])
            makeSize(R[x]);
        size[x] = 1;
        if (L[x])
            size[x] += size[L[x]];
        if (R[x])
            size[x] += size[R[x]];
    }
}
void keepSize(int x) {
    while (x) {
        size[x] = 1;
        if (L[x])
            size[x] += size[L[x]];
        if (R[x])
            size[x] += size[R[x]];
        x = P[x];
    }
}
int insert(T k) {
    int z;
    if (Top) {
        z = bin[--Top];
    } else {
        z = ++C;
        pri[z] = rand();
    }
    L[z] = R[z] = P[z] = 0;
    key[z] = k;
    int x = root, y = 0;
    while (x) {
        y = x;
        if (k < key[x])
            x = L[x];
        else
            x = R[x];
    }
    P[z] = y;
    if (!y)
        root = z;
    else if (k < key[y])
```

```
        L[y] = z;
    else
        R[y] = z;

    while (P[z] && pri[z] < pri[P[z]]) {
        if (z == L[P[z]])
            rightRotate(P[z]);
        else
            leftRotate(P[z]);
    }
    if (!P[z])
        root = z;

    return z;
}
void del(int z) {
    int x, y;
    if (!L[z] || !R[z])
        y = z;
    else
        y = next(z);
    if (L[y])
        x = L[y];
    else
        x = R[y];
    if (x)
        P[x] = P[y];
    if (!P[y])
        root = x;
    else if (y == L[P[y]])
        L[P[y]] = x;
    else
        R[P[y]] = x;
    if (y != z)
        key[z] = key[y];

    bin[Top++] = y;
}
int search(T& k) {
    int x = root;
    while (x && !(key[x] == k)) {
        if (k < key[x])
            x = L[x];
        else
            x = R[x];
    }
    return x;
}
int select(int x, int i) {
    if (!x)
        return 0;
    int r = size[L[x]] + 1;
    if (i == r)
        return x;
    else if (i < r)
        return select(L[x], i);
    else
        return select(R[x], i - r);
}
int select(int i) {
    return select(root, i);
}
int rank(int x) {
```

```
        int r = size[L[x]] + 1;
        int y = x;
        while (y != root) {
            if (y == R[P[y]])
                r += size[L[P[y]]] + 1;
            y = P[y];
        }
        return r;
    }
    double findBest(double k) {
        int x = root;
        while (x) {
            if (sig(k-key[x].left) <= 0 && sig(key[x].right-k) <= 0)
                return key[x].y - k * key[x].x;
            else if (sig(key[x].right-k) < 0)
                x = L[x];
            else
                x = R[x];
        }
    }
};
Treap<Queue> tree;
int N;
double S, dp[100010];
struct Data {
    double A, B, R;
};
Data a[100010];
void init() {
    for (int i = 1; i <= N; i++) {
        a[i].A = nextDouble();
        a[i].B = nextDouble();
        a[i].R = nextDouble();
    }
}
int isRight(Queue a, Queue b, Queue c) {
    Queue p, q;
    p.x = b.x - a.x;
    p.y = b.y - a.y;
    q.x = c.x - b.x;
    q.y = c.y - b.y;
    return sig(p.x*q.y-q.x*p.y) < 0;
}
void update(Queue current) {
    int t = tree.search(current);
    if (!t) {
        t = tree.insert(current);
        int r, s;
        r = tree.pre(t);
        s = tree.next(t);
        if (r && s) {
            if (!isRight(tree.key[r], tree.key[t], tree.key[s])) {
                tree.del(t);
                return;
            }
        }
    } else {
        if (sig(current.y-tree.key[t].y) <= 0)
            return;
        else {
            tree.del(t);
            t = tree.insert(current);
        }
```

```
    }
    Queue temp = tree.key[t];
    int p, q;
    //left
    p = tree.pre(t);
    while (p) {
        q = tree.pre(p);
        if (q && !isRight(tree.key[q], tree.key[p], current)) {
            tree.del(p);
            t = tree.search(temp);
        } else
            break;
        p = tree.pre(t);
    }
    p = tree.pre(t);
    if (p) {
        tree.key[t].left = (tree.key[p].y - tree.key[t].y) /
(tree.key[p].x
                - tree.key[t].x);
        tree.key[p].right = tree.key[t].left;
    } else {
        tree.key[t].left = 0x7FFFFFFF;
    }
    //right
    p = tree.next(t);
    while (p) {
        q = tree.next(p);
        if (q && !isRight(current, tree.key[p], tree.key[q])) {
            tree.del(p);
            t = tree.search(temp);
        } else
            break;
        p = tree.next(t);
    }
    p = tree.next(t);
    if (p) {
        tree.key[t].right = (tree.key[p].y - tree.key[t].y) /
(tree.key[p].x
                - tree.key[t].x);
        tree.key[p].left = tree.key[t].right;
    } else {
        tree.key[t].right = -0x7FFFFFFF;
    }
}
void work() {
    double result = S;
    dp[1] = S;
    Queue current;
    for (int i = 2; i <= N; i++) {
        dp[i] = dp[i - 1];
        current.y = dp[i - 1] / (a[i - 1].A * a[i - 1].R + a[i -
1].B);
        current.x = (dp[i - 1] - a[i - 1].B * current.y) / a[i - 1].A;
        update(current);
        dp[i] = max(dp[i], tree.findBest(-a[i].A / a[i].B) * a[i].B);
        result = max(result, dp[i]);
    }
    printf("%.3lf\n", result);
}
```

查找树决策的单调队列DP

```
struct Data {
```

```
    int index;
    long long value;
};
long long a[100010], sum[100010], dp[100010];
int b[100010];
Data queue[500010];
struct State {
    int firstIndex, lastIndex;
    long long value;
    bool operator <(State p) {
        return value < p.value;
    }
    bool operator >(State p) {
        return value > p.value;
    }
    bool operator ==(State p) {
        return value == p.value;
    }
    void build() {
        value = dp[firstIndex] + a[lastIndex];
    }
};
int N;
long long M;
typedef struct Node {
    struct Node *parent, *left, *right;
    int pri;
    State key;
} TreeNode;
TreeNode *root = NULL;
void rightRotate(TreeNode *x) {
    TreeNode *y = x->left;
    x->left = y->right;
    if (y->right)
        y->right->parent = x;
    y->parent = x->parent;
    if (!x->parent)
        root = y;
    else if (x == x->parent->left)
        x->parent->left = y;
    else
        x->parent->right = y;
    y->right = x;
    x->parent = y;
}
void leftRotate(TreeNode *x) {
    TreeNode *y = x->right;
    x->right = y->left;
    if (y->left)
        y->left->parent = x;
    y->parent = x->parent;
    if (!x->parent)
        root = y;
    else if (x == x->parent->left)
        x->parent->left = y;
    else
        x->parent->right = y;
    y->left = x;
    x->parent = y;
}
TreeNode* min(TreeNode *x) {
    while (x && x->left)
        x = x->left;
```

```
    return x;
}
TreeNode* max(TreeNode *x) {
    while (x && x->right)
        x = x->right;
    return x;
}
TreeNode* suc(TreeNode *x) {
    if (x->right)
        return min(x->right);
    TreeNode *y = x->parent;
    while (y && x == y->right) {
        x = y;
        y = y->parent;
    }
    return y;
}
void insert(State k) {
    TreeNode *z = new TreeNode;
    z->pri = rand();
    z->left = z->right = z->parent = NULL;
    z->key = k;
    TreeNode *x = root, *y = NULL;
    while (x) {
        y = x;
        if (k < x->key)
            x = x->left;
        else
            x = x->right;
    }
    z->parent = y;
    if (!y)
        root = z;
    else if (k < y->key)
        y->left = z;
    else
        y->right = z;
    //Treap
    while (z->parent && z->pri < z->parent->pri) {
        if (z == z->parent->left)
            rightRotate(z->parent);
        else
            leftRotate(z->parent);
    }
    if (!z->parent)
        root = z;
}
void del(TreeNode *z) {
    if (!z)
        return;
    TreeNode *x, *y;
    if (!z->left || !z->right)
        y = z;
    else
        y = suc(z);
    if (y->left)
        x = y->left;
    else
        x = y->right;
    if (x)
        x->parent = y->parent;
    if (!y->parent)
        root = x;
```

```
    else if (y == y->parent->left)
        y->parent->left = x;
    else
        y->parent->right = x;
    if (y != z)
        z->key = y->key;
    delete y;
}
TreeNode* search(TreeNode *x, State k) {
    while (x && !(k == x->key)) {
        if (k < x->key)
            x = x->left;
        else
            x = x->right;
    }
    return x;
}
void init() {
    root = NULL;
    memset(dp, 0, sizeof(dp));
    memset(b, 0, sizeof(b));
    sum[0] = 0;
    for (int i = 1; i <= N; i++) {
        scanf("%I64d", a + i);
        sum[i] = sum[i - 1] + a[i];
        int low = 1, high = i, mid = (low + high) / 2;
        while (low <= high) {
            if (sum[i] - sum[mid - 1] <= M) {
                b[i] = mid;
                high = mid - 1;
                mid = (low + high) / 2;
            } else {
                low = mid + 1;
                mid = (low + high) / 2;
            }
        }
    }
}
void work() {
    Data temp;
    int head = 0, tail = -1;
    queue[++tail].index = 1, queue[tail].value = a[1];
    dp[1] = a[1];
    for (int i = 1; i <= N; i++) {
        if (a[i] > M) {
            printf("-1\n");
            return;
        }
        temp.index = i;
        temp.value = a[i];
        while (head <= tail && queue[tail].value <= a[i]) {
            if (tail != head) {
                State tmp;
                tmp.value = dp[queue[tail - 1].index] +
queue[tail].value;
                del(search(root, tmp));
            }
            tail--;
        }
        while (head <= tail && queue[head].index < b[i] - 1) {
            if (tail != head) {
                State tem;
                tem.value = dp[queue[head].index] + queue[head +
```

```
1].value;
                del(search(root, tem));
            }
            head++;
        }
        queue[++tail] = temp;
        if (head != tail) {
            State tem;
            tem.value = dp[queue[tail - 1].index] + a[i];
            insert(tem);
        }
        TreeNode *tmp = min(root);
        dp[i] = dp[b[i] - 1] + queue[head].value;
        if (tmp && tmp->key.value < dp[i])
            dp[i] = tmp->key.value;
    }
    printf("%I64d\n", dp[N]);
}
```

精确覆盖 Dancing Links

```
const int WID = 1010;
const int HGT = 1010;
const int SIZE = WID * (HGT + 1) + 10;
int arr[WID][HGT], cnt[WID];
struct Dancer {
#define Max 0x7FFFFFFF
    int L[SIZE], R[SIZE], U[SIZE], D[SIZE], C[SIZE], Row[SIZE];
    int S[WID + 10];
    int width, height, t;
    void init(int width, int height) {
        this->width = width;
        this->height = height;
        int p, x, y, last;
        for (x = 1; x <= width; x++) {
            L[x] = x - 1;
            R[x] = x + 1;
            U[x] = D[x] = x;
            S[x] = 0;
        }
        R[width] = 0;
        p = width + 1;
        for (y = 1; y <= height; y++) {
            last = R[0] = L[0] = 0;
            for (t = 1; t <= cnt[y]; t++) {
                int x = arr[y][t];
                U[p] = U[x];
                C[p] = D[p] = x;
                L[p] = last;
                S[x]++;
                Row[p] = y;
                last = R[last] = U[x] = D[U[x]] = p++;
            }
            R[last] = R[0];
            L[R[0]] = last;
        }
        L[0] = width;
        R[0] = 1;
        S[0] = Max;
    }
    void remove(const int &c) {
        int i, j;
        L[R[c]] = L[c];
```

```
        R[L[c]] = R[c];
        for (i = D[c]; i != c; i = D[i]) {
            for (j = R[i]; j != i; j = R[j]) {
                U[D[j]] = U[j];
                D[U[j]] = D[j];
                --S[C[j]];
            }
        }
    }
    void resume(const int &c) {
        int i, j;
        for (i = U[c]; i != c; i = U[i]) {
            for (j = L[i]; j != i; j = L[j]) {
                ++S[C[j]];
                U[D[j]] = j;
                D[U[j]] = j;
            }
        }
        L[R[c]] = c;
        R[L[c]] = c;
    }
    bool dance() {
        if (R[0] == 0)
            return true;
        int c = 0, i, j;
        for (i = R[0]; i != 0/* && i<=N*/; i = R[i])
            if (S[i] < S[c])
                c = i;
        //if (!c) return true;  控制拿前N列
        remove(c);
        for (i = D[c]; i != c; i = D[i]) {
            for (j = R[i]; j != i; j = R[j])
                remove(C[j]);
            //result[deep++]=Row[i];   记录解
            if (dance())
                return true;
            //deep--;
            for (j = L[i]; j != i; j = L[j])
                resume(C[j]);
        }
        resume(c);
        return false;
    }
};
```

重复覆盖 Dancing Links

```
const int WID = 1010;
const int HGT = 1010;
const int SIZE = WID * (HGT + 1) + 10;
int arr[WID][HGT], cnt[WID];
int best = 0x7FFFFFFF;
struct Dancer {
#define Max 0x7FFFFFFF
    int L[SIZE], R[SIZE], U[SIZE], D[SIZE], C[SIZE], Row[SIZE];
    int S[WID + 10];
    int width, height;
    void init(int width, int height) {
        this->width = width;
        this->height = height;
        int p, x, y, last, t;
        for (x = 1; x <= width; x++) {
```

```
            L[x] = x - 1;
            R[x] = x + 1;
            U[x] = D[x] = x;
            S[x] = 0;
        }
        R[width] = 0;
        p = width + 1;
        for (y = 1; y <= height; y++) {
            last = R[0] = L[0] = 0;
            for (t = 1; t <= cnt[y]; t++) {
                int x = arr[y][t];
                U[p] = U[x];
                C[p] = D[p] = x;
                L[p] = last;
                S[x]++;
                Row[p] = y;
                last = R[last] = U[x] = D[U[x]] = p++;
            }
            R[last] = R[0];
            L[R[0]] = last;
        }
        L[0] = width;
        R[0] = 1;
        S[0] = Max;
    }
    void remove(const int &c) {
        int i;
        for (i = D[c]; i != c; i = D[i]) {
            L[R[i]] = L[i];
            R[L[i]] = R[i];
        }
    }
    void resume(const int &c) {
        int i;
        for (i = U[c]; i != c; i = U[i]) {
            L[R[i]] = i;
            R[L[i]] = i;
        }
    }
    int h() {
        bool hash[51];
        memset(hash, false, sizeof(hash));
        int ret = 0;
        for (int c = R[0]; c != 0; c = R[c]) {
            if (!hash[c]) {
                ret++;
                hash[c] = true;
                for (int i = D[c]; i != c; i = D[i]) {
                    for (int j = R[i]; j != i; j = R[j]) {
                        hash[C[j]] = true;
                    }
                }
            }
        }
        return ret;
    }
    bool dance(int deep) {
        if (deep + h() > best)
            return false;
        if (R[0] == 0) {
            best = deep;
            return true;
        }
```

```
        int c = 0, i, j;
        bool flag = false;
        for (i = R[0]; i != 0; i = R[i])
            if (S[i] < S[c])
                c = i;
        for (i = D[c]; i != c; i = D[i]) {
            remove(i);
            for (j = R[i]; j != i; j = R[j]) {
                remove(j);
            }
            if (dance(deep + 1, lim))
                flag = true;
            for (j = L[i]; j != i; j = L[j]) {
                resume(j);
            }
            resume(i);
        }
        return flag;
    }
};
```