LIS

```
int lis_size(vector<int>& nums) {
    vector<int> tail;
    for (auto x : nums) {
        auto it = lower_bound(tail.begin(), tail.end(), x);
        if (it == tail.end()) tail.push_back(x);
        else *it = x;
    }
    return tail.size();
}

int lis_with_sequence(const vector<int>& a,
                          vector<int>& out_seq)
    {
        int n = a.size();
        vector<int> tail;
        vector<int> tail_idx;
        vector<int> prev(n, -1);
        tail.reserve(n);
        tail_idx.reserve(n);

        for (int i = 0; i < n; ++i) {
            int x = a[i];
            auto it = lower_bound(tail.begin(), tail.end(), x);
            int pos = it - tail.begin();
            if (it == tail.end()) {
                tail.push_back(x);
                tail_idx.push_back(i);
            }
            else {
                *it = x;
                tail_idx[pos] = i;
            }
            if (pos > 0)
                prev[i] = tail_idx[pos-1];
        }

        int lis_len = tail.size();
        out_seq.clear();
        for (int cur = tail_idx[lis_len - 1]; cur >= 0; cur = prev[cur])
            out_seq.push_back(a[cur]);
        reverse(out_seq.begin(), out_seq.end());

        return lis_len;
    }
```

0/1 Knapsack

```
int knapsack(int W, vector<int> &val, vector<int> &wt) {
    vector<int> dp(W + 1, 0);
    for (int i = 1; i <= wt.size(); i++)
        for (int j = W; j >= wt[i - 1]; j--)
            dp[j] = max(dp[j], dp[j - wt[i - 1]] + val[i - 1]);


    return dp[W];
}

int unbounded_knapSack(int capacity, vector<int> &val, vector<int> &wt) {
    vector<int> dp(capacity + 1, 0);
    for (int i = val.size() - 1; i >= 0; i--) {
        for (int j = 1; j <= capacity; j++) {
            int take = 0;
            if (j - wt[i] >= 0) {
                take = val[i] + dp[j - wt[i]];
            }
            int noTake = dp[j];
            dp[j] = max(take, noTake);
        }
    }
    return dp[capacity];
}
```

edit distance

```
string a, b; cin >> a >> b;
int na = a.size(), nb = b.size();
int dp[na+1][nb+1];
memset(dp, 0, sizeof dp);

for (int i = 1; i <= na; i++)
    dp[i][0] = i;
for (int j = 1; j <= nb; j++)
    dp[0][j] = j;
for (int i = 1; i <= na; i++) {
    for (int j = 1; j <= nb; j++) {
        dp[i][j] = min({dp[i-1][j], dp[i][j-1], dp[i-1][j-1]}) + 1;
        if (a[i-1] == b[j-1])
            chmin(dp[i][j], dp[i-1][j-1]);
    }
}
cout << dp[na][nb];
```

remove game

```
void solve() {
    int n; cin >> n;
    vector<int> v(n);
    getv(v);
    ll diff[n+1][n+1]{}; // mx diff p1-p2 on interval [l, r]
    for (int l = n-1; ~l; l--) {
        for (int r = 0; r < n; r++) {
            // dp[l][r] = mx( v[l]-dp[l+1][r], v[r]-dp[l][r-1] )
            if (l == r)
                diff[l][r] = v[l];
            else diff[l][r] = max<ll>(v[l] - diff[l+1][r], v[r]-diff[l][r-
1]);
        }
    }

    cout << (accumulate(all(v), 0ll) + diff[0][n-1])/2;
}
```

count subsets sum to k

```
int perfectSum(vector<int> &arr, int target) {
    int n = arr.size();
    vector<int> dp(target + 1, 0), ndp(target + 1, 0);
    dp[0] = 1;
    for (int i = 1; i <= n; i++) {
        ndp = dp;
        for (int j = 0; j <= target; j++) {
            if (j >= arr[i - 1]) {
                ndp[j] += dp[j - arr[i - 1]];
            }
        }
        dp = ndp;
    }
    return ndp[target];
}
```

longest common subsequence

```
void lcs(char* X, char* Y, int m, int n)
{
    int L[m + 1][n + 1];
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0 || j == 0)
                L[i][j] = 0;
```

```
            else if (X[i - 1] == Y[j - 1])
                L[i][j] = L[i - 1][j - 1] + 1;
            else
                L[i][j] = max(L[i - 1][j], L[i][j - 1]);
        }
    }
    int index = L[m][n];

    char lcs[index + 1];
    lcs[index] = '\0';
    int i = m, j = n;
    while (i > 0 && j > 0) {
        if (X[i - 1] == Y[j - 1]) {
            lcs[index - 1]
                = X[i - 1];
            i--;
            j--;
            index--;
        }
        else if (L[i - 1][j] > L[i][j - 1])
            i--;
        else
            j--;
    }
}
```

shortest common supersequence

```
while (i > 0 && j > 0) {
    if (str1[i - 1] == str2[j - 1]) {
        result.push_back(str1[i - 1]);
        i--;
        j--;
    } else if (dp[i - 1][j] > dp[i][j - 1]) {
        result.push_back(str1[i - 1]);
        i--;
    } else {
        result.push_back(str2[j - 1]);
        j--;
    }
}
while (i > 0) {
    result.push_back(str1[i - 1]);
    i--;
}
while (j > 0) {
    result.push_back(str2[j - 1]);
    j--;
}
```

```
reverse(result.begin(), result.end());
return result;
```

hamiltonian paths

```
int rec(int u, int vis) {
    vis |= (1 << u);
    if (u == n-1)
        return __builtin_popcount(vis) == n;

    int& ans = dp[vis][u];
    if (~ans) return ans;
    // dpid[u][vis] = 2;
    ans = 0;
    each(ver, g[u])
        if (!((vis >> ver) & 1))
            ans = (ans + rec(ver, vis | (1 << ver)))%mod;
    return ans;
}
```

hamiltonian paths


```
int rec(int u, int vis) {
    vis |= (1 << u);
    if (u == n-1)
        return __builtin_popcount(vis) == n;

    int& ans = dp[vis][u];
    if (~ans) return ans;
    // dpid[u][vis] = 2;
    ans = 0;
    each(ver, g[u])
        if (!((vis >> ver) & 1))
            ans = (ans + rec(ver, vis | (1 << ver)))%mod;
    return ans;
}
```