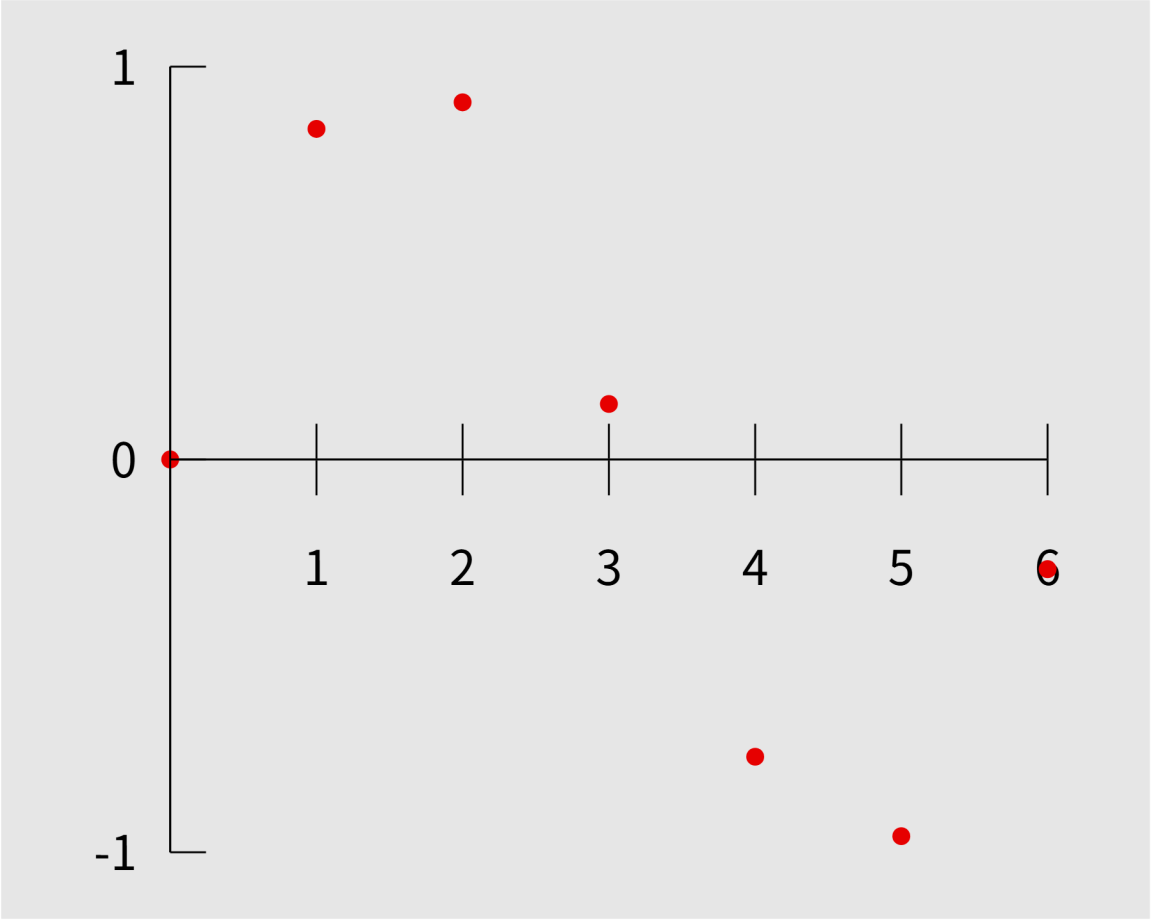
# 插值

## 引入

插值是一种通过已知的、离散的数据点推算一定范围内的新数据点的方法。插值法常用于函数拟合中。

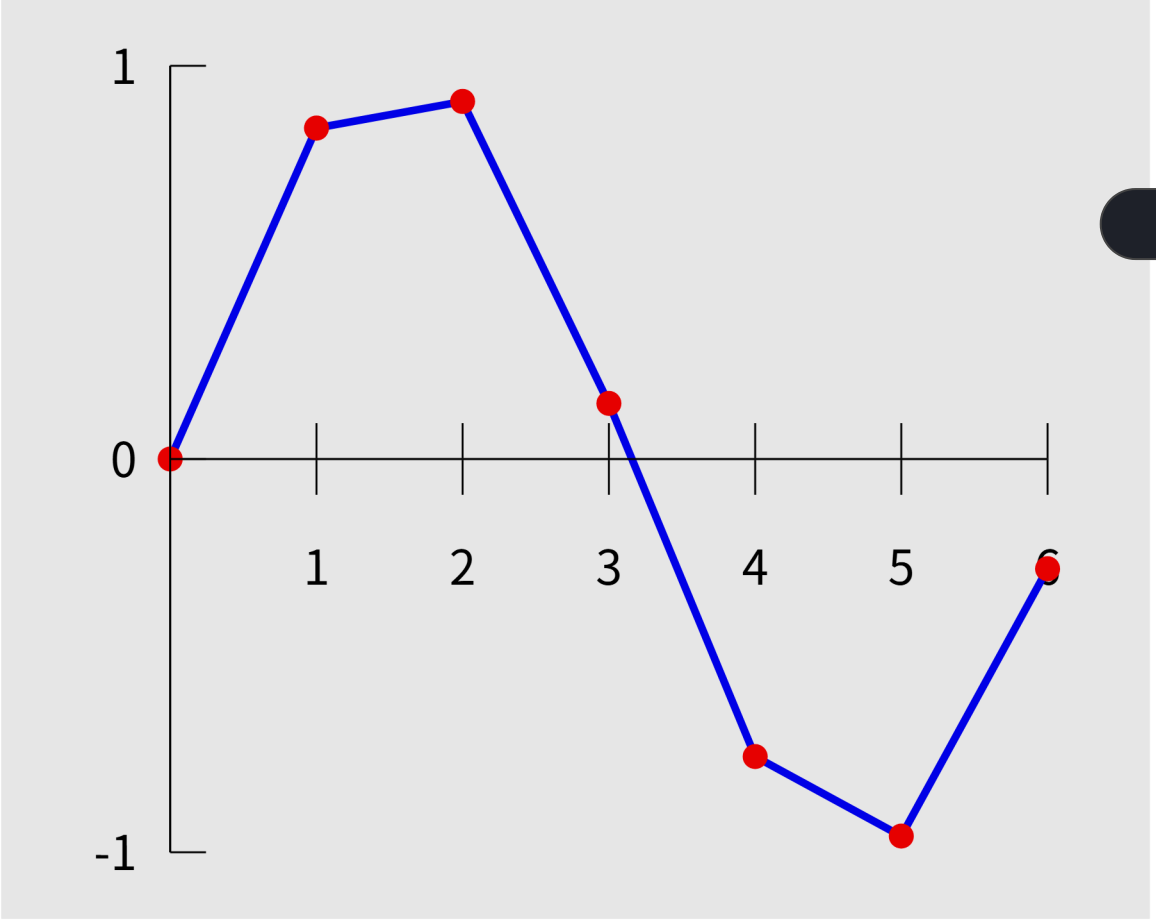
例如对数据点：

| $x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| $f(x)$ | 0 | 0.8415 | 0.9093 | 0.1411 | $-0.7568$ | $-0.9589$ | $-0.2794$ |



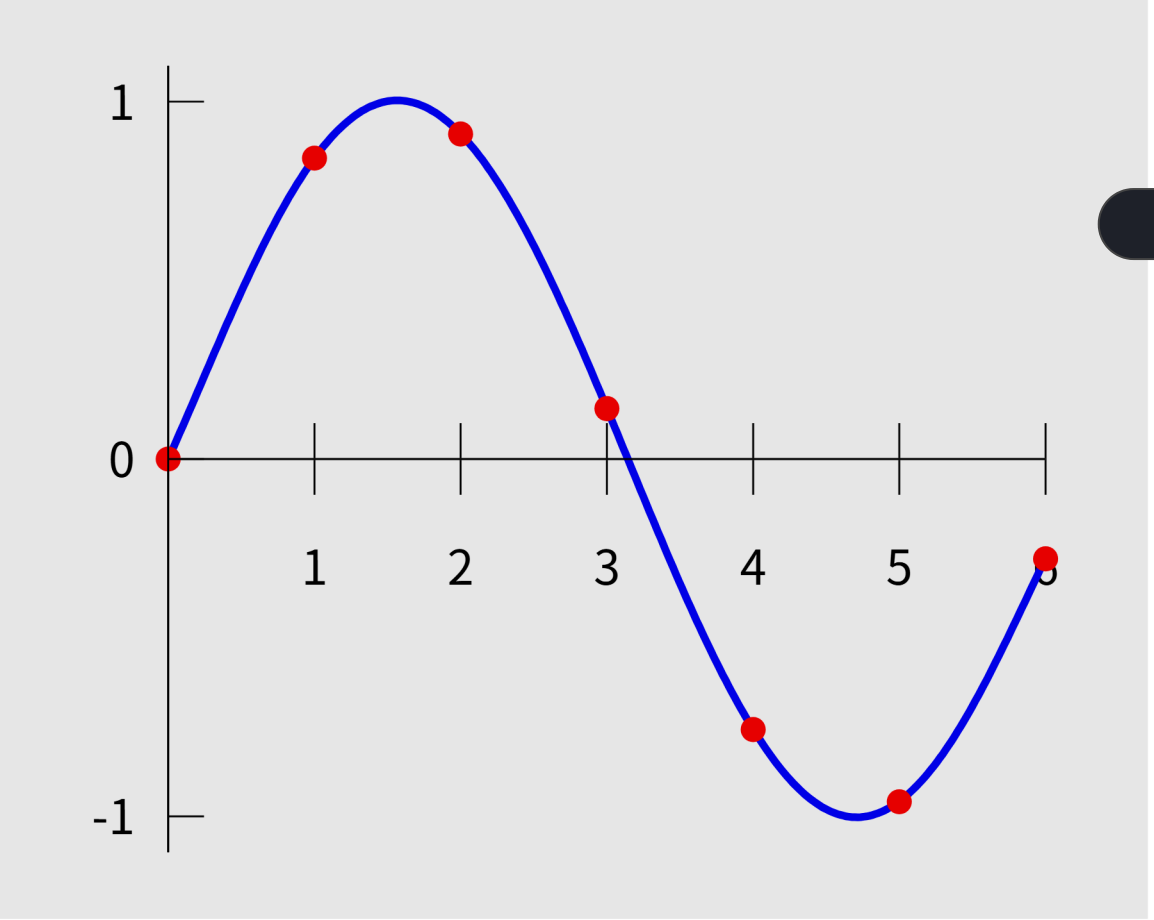

其中 $f(x)$ 未知，插值法可以通过按一定形式拟合 $f(x)$ 的方式估算未知的数据点。

例如，我们可以用分段线性函数拟合 $f(x)$：




这种插值方式叫做 线性插值。

我们也可以用多项式拟合 $f(x)$：




这种插值方式叫做 多项式插值。

多项式插值的一般形式如下：

**多项式插值**

对已知的 $n+1$ 的点 $(x_0,y_0),(x_1,y_1),\dots,(x_n,y_n)$，求 $n$ 阶多项式 $f(x)$ 满足

$$f(x_i)=y_i,\qquad \forall i=0,1,\dots,n$$

下面介绍多项式插值中的两种方式：Lagrange 插值法与 Newton 插值法。不难证明这两种方法得到的结果是相等的。

## Lagrange 插值法

由于要求构造一个函数 $f(x)$ 过点 $P_1(x_1,y_1),P_2(x_2,y_2),\cdots,P_n(x_n,y_n)$. 首先设第 $i$ 个点在 $x$ 轴上的投影为 $P_i'(x_i,0)$.

考虑构造 $n$ 个函数 $f_1(x),f_2(x),\cdots,f_n(x)$，使得对于第 $i$ 个函数 $f_i(x)$，其图像过 $\begin{cases}P_j'(x_j,0),(j\neq i)\\P_i(x_i,y_i)\end{cases}$，则可知题目所求的函数 $f(x)=\sum\limits_{i=1}^{n}f_i(x)$.

那么可以设 $f_i(x) = a \cdot \prod_{j \neq i}(x - x_j)$，将点 $P_i(x_i, y_i)$ 代入可以知道 $a = \dfrac{y_i}{\prod_{j \neq i}(x_i - x_j)}$，所以

$$f_i(x) = y_i \cdot \frac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq i}(x_i - x_j)} = y_i \cdot \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

那么我们就可以得出 Lagrange 插值的形式为：

$$f(x) = \sum_{i=1}^{n} y_i \cdot \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

朴素实现的时间复杂度为 $O(n^2)$，可以优化到 $O(n \log^2 n)$，参见 多项式快速插值。

## Luogu P4781【模板】拉格朗日插值

给出 $n$ 个点对 $(x_i, y_i)$ 和 $k$，且 $\forall i, j$ 有 $i \neq j \iff x_i \neq x_j$ 且 $f(x_i) \equiv y_i \pmod{998244353}$ 和 $\deg(f(x)) < n$（定义 $\deg(0) = -\infty$），求 $f(k) \bmod 998244353$.

### 题解

本题中只用求出 $f(k)$ 的值，所以在计算上式的过程中直接将 $k$ 代入即可；有时候则需要进行多次求值等等更为复杂的操作，这时候需要求出 $f$ 的各项系数。代码给出了一种求出系数的实现。

$$f(k) = \sum_{i=1}^{n} y_i \prod_{j \neq i} \frac{k - x_j}{x_i - x_j}$$

本题中，还需要求解逆元。如果先分别计算出分子和分母，再将分子乘进分母的逆元，累加进最后的答案，时间复杂度的瓶颈就不会在求逆元上，时间复杂度为 $O(n^2)$。

因为在固定模 $998244353$ 意义下运算，计算乘法逆元的时间复杂度我们在这里暂且认为是常数时间。

**代码实现**

```cpp
#include <iostream>
#include <vector>

constexpr int MOD = 998244353;
using LL = long long;

int inv(int k) {
  int res = 1;
  for (int e = MOD - 2; e; e /= 2) {
    if (e & 1) res = (LL)res * k % MOD;
    k = (LL)k * k % MOD;
  }
  return res;
}

// 返回 f 满足 f(x_i) = y_i
// 不考虑乘法逆元的时间，显然为 O(n^2)
std::vector<int> lagrange_interpolation(const std::vector<int>
&x,
                                        const std::vector<int>
&y) {
  const int n = x.size();
  std::vector<int> M(n + 1), px(n, 1), f(n);
  M[0] = 1;
  // 求出 M(x) = prod_(i=0..n-1)(x - x_i)
  for (int i = 0; i < n; ++i) {
    for (int j = i; j >= 0; --j) {
      M[j + 1] = (M[j] + M[j + 1]) % MOD;
      M[j] = (LL)M[j] * (MOD - x[i]) % MOD;
    }
  }
  // 求出 px_i = prod_(j=0..n-1, j!=i) (x_i - x_j)
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j)
      if (i != j) {
        px[i] = (LL)px[i] * (x[i] - x[j] + MOD) % MOD;
      }
  }
  // 组合出 f(x) = sum_(i=0..n-1)(y_i / px_i)(M(x) / (x -
x_i))
  for (int i = 0; i < n; ++i) {
    LL t = (LL)y[i] * inv(px[i]) % MOD, k = M[n];
    for (int j = n - 1; j >= 0; --j) {
      f[j] = (f[j] + k * t) % MOD;
      k = (M[j] + k * x[i]) % MOD;
    }
  }
  return f;
}
```

```cpp

int main() {
  std::ios::sync_with_stdio(false);
  std::cin.tie(nullptr);
  int n, k;
  std::cin >> n >> k;
  std::vector<int> x(n), y(n);
  for (int i = 0; i < n; ++i) std::cin >> x[i] >> y[i];
  const auto f = lagrange_interpolation(x, y);
  int v = 0;
  for (int i = n - 1; i >= 0; --i) v = ((LL)v * k + f[i]) %
MOD;
  std::cout << v << '\n';
  return 0;
}
```

## 横坐标是连续整数的 Lagrange 插值

如果已知点的横坐标是连续整数，我们可以做到 $O(n)$ 插值。

设要求 $n$ 次多项式为 $f(x)$，我们已知 $f(1),\cdots,f(n+1)$（$1\le i\le n+1$），考虑代入上面的插值公式：

$$\begin{aligned}f(x)&=\sum_{i=1}^{n+1}y_i\prod_{j\neq i}\frac{x-x_j}{x_i-x_j}\\&=\sum_{i=1}^{n+1}y_i\prod_{j\neq i}\frac{x-j}{i-j}\end{aligned}$$

后面的累乘可以分子分母分别考虑，不难得到分子为：

$$\frac{\prod_{j=1}^{n+1}(x-j)}{x-i}$$

分母的 $i-j$ 累乘可以拆成两段阶乘来算：

$$(-1)^{n+1-i}\cdot(i-1)!\cdot(n+1-i)!$$

于是横坐标为 $1,\cdots,n+1$ 的插值公式：

$$f(x)=\sum_{i=1}^{n+1}(-1)^{n+1-i}y_i\cdot\frac{\prod_{j=1}^{n+1}(x-j)}{(i-1)!(n+1-i)!(x-i)}$$

预处理 $(x-i)$ 前后缀积、阶乘阶乘逆，然后代入这个式子，复杂度为 $O(n)$.

**例题 CF622F The Sum of the k-th Powers**

给出 $n, k$，求 $\sum\limits_{i=1}^{n} i^k$ 对 $10^9+7$ 取模的值。

**题解**

本题中，答案是一个 $k+1$ 次多项式，因此我们可以线性筛出 $1^i, \cdots, (k+2)^i$ 的值然后进行 $O(n)$ 插值。

也可以通过组合数学相关知识由差分法的公式推得下式：

$$f(x)=\sum_{i=1}^{n+1}\binom{x-1}{i-1}\sum_{j=1}^{i}(-1)^{i+j}\binom{i-1}{j-1}y_j=\sum_{i=1}^{n+1}y_i\cdot\frac{\prod\limits_{j=1}^{n+1}(x-j)}{(x-i)\cdot(-1)^{n+1-i}\cdot(i-1)!\cdot(n+1-i)!}$$

**代码实现**

```cpp
// By: Luogu@rui_er(122461)
#include <iostream>
using namespace std;
constexpr int N = 1e6 + 5, mod = 1e9 + 7;

int n, k, tab[N], p[N], pcnt, f[N], pre[N], suf[N], fac[N],
inv[N], ans;

int qpow(int x, int y) {
  int ans = 1;
  for (; y; y >>= 1, x = 1LL * x * x % mod)
    if (y & 1) ans = 1LL * ans * x % mod;
  return ans;
}

void sieve(int lim) {
  f[1] = 1;
  for (int i = 2; i <= lim; i++) {
    if (!tab[i]) {
      p[++pcnt] = i;
      f[i] = qpow(i, k);
    }
    for (int j = 1; j <= pcnt && 1LL * i * p[j] <= lim; j++) {
      tab[i * p[j]] = 1;
      f[i * p[j]] = 1LL * f[i] * f[p[j]] % mod;
      if (!(i % p[j])) break;
    }
  }
  for (int i = 2; i <= lim; i++) f[i] = (f[i - 1] + f[i]) %
mod;
}

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  cin >> n >> k;
  sieve(k + 2);
  if (n <= k + 2) return cout << f[n], 0;
  pre[0] = suf[k + 3] = 1;
  for (int i = 1; i <= k + 2; i++) pre[i] = 1LL * pre[i - 1] *
(n - i) % mod;
  for (int i = k + 2; i >= 1; i--) suf[i] = 1LL * suf[i + 1] *
(n - i) % mod;
  fac[0] = inv[0] = fac[1] = inv[1] = 1;
  for (int i = 2; i <= k + 2; i++) {
    fac[i] = 1LL * fac[i - 1] * i % mod;
    inv[i] = 1LL * (mod - mod / i) * inv[mod % i] % mod;
  }
  for (int i = 2; i <= k + 2; i++) inv[i] = 1LL * inv[i - 1] *
inv[i] % mod;
```

```
    for (int i = 1; i <= k + 2; i++) {
      int P = 1LL * pre[i - 1] * suf[i + 1] % mod;
      int Q = 1LL * inv[i - 1] * inv[k + 2 - i] % mod;
      int mul = ((k + 2 - i) & 1) ? -1 : 1;
      ans = (ans + 1LL * (Q * mul + mod) % mod * P % mod * f[i]
% mod) % mod;
    }
    cout << ans << '\n';
    return 0;
}
```

## Newton 插值法

Newton 插值法是基于高阶差分来插值的方法，优点是支持 $O(n)$ 插入新数据点。

为了实现 $O(n)$ 插入新数据点，我们令：

$$f(x)=\sum_{j=0}^{n}a_jn_j(x)$$

其中 $n_j(x):=\prod_{i=0}^{j-1}(x-x_i)$ 称为 **Newton 基**（Newton basis）。

若解出 $a_j$，则可得到 $f(x)$ 的插值多项式。我们按如下方式定义 **前向差商**（forward divided differences）：

$$\begin{aligned}[y_k]&:=y_k, &&k=0,\dots,n,\\ [y_k,\dots,y_{k+j}]&:=\frac{[y_{k+1},\dots,y_{k+j}]-[y_k,\dots,y_{k+j-1}]}{x_{k+j}-x_k}, &&k=0,\dots,n-j,\ j=1,\dots,n.\end{aligned}$$

则：

$$\begin{aligned}f(x)&=[y_0]+[y_0,y_1](x-x_0)+\cdots+[y_0,\dots,y_n](x-x_0)\dots(x-x_{n-1})\\&=\sum_{j=0}^{n}[y_0,\dots,y_j]n_j(x)\end{aligned}$$

此即 Newton 插值的形式。朴素实现的时间复杂度为 $O(n^2)$.

若样本点是等距的（即 $x_i=x_0+ih$，$i=1,\dots,n$），令 $x=x_0+sh$，Newton 插值的公式可化为：

$$f(x)=\sum_{j=0}^{n}\binom{s}{j}j!h^j[y_0,\dots,y_j]$$

上式称为 **Newton 前向差分公式**（Newton forward divided difference formula）。

**Note**

若样本点是等距的，我们还可以推出：

$$[y_k, \dots, y_{k+j}] = \frac{1}{j! h^j} \Delta^{(j)} y_k$$

其中 $\Delta^{(j)} y_k$ 为 **前向差分**（forward differences），定义如下：

$$\begin{aligned} \Delta^{(0)} y_k &:= y_k, && k = 0, \dots, n, \\ \Delta^{(j)} y_k &:= \Delta^{(j-1)} y_{k+1} - \Delta^{(j-1)} y_k, && k = 0, \dots, n-j,\ j = 1, \dots, n. \end{aligned}$$

**代码实现（Luogu P4781【模板】拉格朗日插值）**

```cpp
#include <cstdint>
#include <iostream>
#include <vector>
using namespace std;

constexpr uint32_t MOD = 998244353;

struct mint {
  uint32_t v_;

  mint() : v_(0) {}

  mint(int64_t v) {
    int64_t x = v % (int64_t)MOD;
    v_ = (uint32_t)(x + (x < 0 ? MOD : 0));
  }

  friend mint inv(mint const &x) {
    int64_t a = x.v_, b = MOD;
    if ((a %= b) == 0) return 0;
    int64_t s = b, m0 = 0;
    for (int64_t q = 0, _ = 0, m1 = 1; a;) {
      _ = s - a * (q = s / a);
      s = a;
      a = _;
      _ = m0 - m1 * q;
      m0 = m1;
      m1 = _;
    }
    return m0;
  }

  mint &operator+=(mint const &r) {
    if ((v_ += r.v_) >= MOD) v_ -= MOD;
    return *this;
  }

  mint &operator-=(mint const &r) {
    if ((v_ -= r.v_) >= MOD) v_ += MOD;
    return *this;
  }

  mint &operator*=(mint const &r) {
    v_ = (uint32_t)((uint64_t)v_ * r.v_ % MOD);
    return *this;
  }

  mint &operator/=(mint const &r) { return *this = *this *
inv(r); }
```

```
  friend mint operator+(mint l, mint const &r) { return l += r;
}

  friend mint operator-(mint l, mint const &r) { return l -= r;
}

  friend mint operator*(mint l, mint const &r) { return l *= r;
}

  friend mint operator/(mint l, mint const &r) { return l /= r;
}
};

template <class T>
struct NewtonInterp {
  // {(x_0,y_0),...,(x_{n-1},y_{n-1})}
  vector<pair<T, T>> p;
  // dy[r][l] = [y_l,...,y_r]
  vector<vector<T>> dy;
  // (x-x_0)...(x-x_{n-1})
  vector<T> base;
  // [y_0]+...+[y_0,y_1,...,y_n](x-x_0)...(x-x_{n-1})
  vector<T> poly;

  void insert(T const &x, T const &y) {
    p.emplace_back(x, y);
    size_t n = p.size();
    if (n == 1) {
      base.push_back(1);
    } else {
      size_t m = base.size();
      base.push_back(0);
      for (size_t i = m; i; --i) base[i] = base[i - 1];
      base[0] = 0;
      for (size_t i = 0; i < m; ++i)
        base[i] = base[i] - p[n - 2].first * base[i + 1];
    }
    dy.emplace_back(p.size());
    dy[n - 1][n - 1] = y;
    if (n > 1) {
      for (size_t i = n - 2; ~i; --i)
        dy[n - 1][i] =
            (dy[n - 2][i] - dy[n - 1][i + 1]) / (p[i].first -
p[n - 1].first);
    }
    poly.push_back(0);
    for (size_t i = 0; i < n; ++i) poly[i] = poly[i] + dy[n - 1]
[0] * base[i];
  }

  T eval(T const &x) {
```

```
    T ans{};
    for (auto it = poly.rbegin(); it != poly.rend(); ++it) ans =
ans * x + *it;
    return ans;
  }
};

int main() {
  NewtonInterp<mint> ip;
  int n, k;
  cin >> n >> k;
  for (int i = 1, x, y; i <= n; ++i) {
    cin >> x >> y;
    ip.insert(x, y);
  }
  cout << ip.eval(k).v_;
  return 0;
}
```

### 横坐标是连续整数的 Newton 插值

例如：求某三次多项式 $f(x)=\sum_{i=0}^{3}a_ix^i$ 的多项式系数，已知 $f(1)$ 至 $f(6)$ 的值分别为 $1,5,14,30,55,91$。

$$\begin{array}{ccccccccccc} 1 & & 5 & & 14 & & 30 & & 55 & & 91 \\ & 4 & & 9 & & 16 & & 25 & & 36 & \\ & & 5 & & 7 & & 9 & & 11 & & \\ & & & 2 & & 2 & & 2 & & & \end{array}$$

第一行为 $f(x)$ 的连续的前 $n$ 项；之后的每一行为之前一行中对应的相邻两项之差。观察到，如果这样操作的次数足够多（前提是 $f(x)$ 为多项式），最终总会返回一个定值。

计算出第 $i-1$ 阶差分的首项为 $\sum_{j=1}^{i}(-1)^{i+j}\binom{i-1}{j-1}f(j)$，第 $i-1$ 阶差分的首项对 $f(k)$ 的贡献为 $\binom{k-1}{i-1}$ 次。

$$f(k)=\sum_{i=1}^{n}\binom{k-1}{i-1}\sum_{j=1}^{i}(-1)^{i+j}\binom{i-1}{j-1}f(j)$$

时间复杂度为 $O(n^2)$.

## C++ 中的实现

自 C++ 20 起，标准库添加了 `std::midpoint` 和 `std::lerp` 函数，分别用于求中点和线性插值。

## 习题

- 「NOIP2020」微信步数
- 「联合省选 2022」填树
- 「NOI2019」机器人

## 参考资料


1. Interpolation - Wikipedia
2. Newton polynomial - Wikipedia


---


本页面最近更新：2025/5/3 19:43:25，更新历史

发现错误？想一起完善？ 在 GitHub 上编辑此页！

本页面贡献者：caibyte, Tiphereth-A, Watersail2005, AtomAlpaca, billchenchina, c-forrest, Chrogeek, Early0v0, EndlessCheng, Enter-tainer, Ghastlcon, Henry-ZHR, hly1204, hsfzLZH1, Ir1d, kenlig, Marcythm, megakite, Peanut-Tang, qwqAutomaton, qz-cqy, StudyingFather, swift-zym, swiftqwq, TrisolarisHD, x4Cx58x54, Xeonacid, xiaopangfeiyu, YanWQ-monad

本页面的全部内容在 **CC BY-SA 4.0 和 SATA** 协议之条款下提供，附加条款亦可能应用