Ho Chi Minh City International University ACM Team Notebook (2012)

Contents

[1. Mathematical Formula 2](#_Toc402266629)

[2. Long Arithmetics (Phép tính số lớn) 5](#_Toc402266630)

[3. Search & Sort 6](#_Toc402266631)

[4. Dynamic Programming 6](#_Toc402266632)

[a. Bad Neighbors (Bài toán DP theo dạng hình tròn). 6](#_Toc402266633)

[b. Matrix Chain Multiplication (Stripe Painter) 6](#_Toc402266634)

[c. Longest Common Subsequence 7](#_Toc402266635)

[d. Longest Common Palindrome Subsequence 7](#_Toc402266636)

[e. Longest Increasing Subsequence 8](#_Toc402266637)

[f. Levenshtein Distance 8](#_Toc402266638)

[g. Knapsack Problem 8](#_Toc402266639)

[h. Subset Sum Problem (TH đặc biệt của Knapsack) 9](#_Toc402266640)

[i. Partition (chia làm 2 phần bằng nhau) 10](#_Toc402266641)

[j. DIV 10](#_Toc402266642)

[k. Tick 11](#_Toc402266643)

[l. RTicket 11](#_Toc402266644)

[m. Ladder 11](#_Toc402266645)

[n. Domino 12](#_Toc402266646)

[o. Winning streak 13](#_Toc402266647)

[5. Dates 13](#_Toc402266648)

[6. Geometry 14](#_Toc402266649)

[a. Formula 14](#_Toc402266650)

[b. Hình Giải Tích 16](#_Toc402266651)

[c. Tiếp Tuyến 17](#_Toc402266652)

[d. Standford Geometry: 18](#_Toc402266653)

[e. Geo3D 22](#_Toc402266654)

[7. Graph 23](#_Toc402266655)

[a. FastDijkstra.cc 23](#_Toc402266656)

[b. Bellman\_Ford 24](#_Toc402266657)

[c. Floyd\_Warshall 25](#_Toc402266658)

[d. Euler\_Path 25](#_Toc402266659)

[e. Hamiltonian\_Path/Circuit 26](#_Toc402266660)

[f. Prim(MSTree) 26](#_Toc402266661)

[g. Tarjan (Liet ke tat ca thanh phan lien thong manh). 27](#_Toc402266662)

[h. Minimum Cost Bipartite Matching: Hungarian algorithm 28](#_Toc402266663)

[i. Max Flow: Ford-Fulkerson algorithm 29](#_Toc402266664)

[j. Maximum Matching 29](#_Toc402266665)

[k. Dinic.cc 31](#_Toc402266666)

[l. MinCostMaxFlow.cc 32](#_Toc402266667)

[m. PushRelabel.cc 33](#_Toc402266668)

[n. MinCut.cc 35](#_Toc402266669)

[o. MinCostMatching.cc 36](#_Toc402266670)

[8. String Processing 39](#_Toc402266671)

[a. KMP 39](#_Toc402266672)

[b. Suffix Array 39](#_Toc402266673)

[9. Appendix 40](#_Toc402266674)

# Mathematical Formula

*// Primes less than 1000:*

*// 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47*

*// 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113*

*// 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197*

*// 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281*

*// 283 293 307 311 313 317 331 337 347 349 353 359 367 373 379*

*// 383 389 397 401 409 419 421 431 433 439 443 449 457 461 463*

*// 467 479 487 491 499 503 509 521 523 541 547 557 563 569 571*

*// 577 587 593 599 601 607 613 617 619 631 641 643 647 653 659*

*// 661 673 677 683 691 701 709 719 727 733 739 743 751 757 761*

*// 769 773 787 797 809 811 821 823 827 829 839 853 857 859 863*

*// 877 881 883 887 907 911 919 929 937 941 947 953 967 971 977*

*// 983 991 997*

*// Catalan Number:*

*//* [*1*](http://en.wikipedia.org/wiki/1_(number)) *1*[*2*](http://en.wikipedia.org/wiki/2_(number))[*5*](http://en.wikipedia.org/wiki/5_(number))[*14*](http://en.wikipedia.org/wiki/14_(number))[*42*](http://en.wikipedia.org/wiki/42_(number))[*132*](http://en.wikipedia.org/wiki/132_(number)) *429 1430 4862 16796 58786 208012 742900 2674440 9694845*

*// 35357670 129644790 477638700 1767263190 6564120420 24466267020 91482563640 343059613650*

*// 1289904147324 4861946401452*

*// 300 số Fibonacci*

// 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946

// 17711 28657 46368 75025 121393 196418 317811 514229 832040 1346269 2178309 3524578

// 5702887 9227465 14930352 24157817 39088169 63245986 102334155 165580141 267914296

// 433494437 701408733 1134903170 1836311903 2971215073 4807526976 7778742049 12586269025

|  |  |  |
| --- | --- | --- |
| **Commonly Used Libraries:**  #include <stdio.h>  #include <stdlib.h>  #include <iostream>  #include <fstream>  #include <sstream> | #include <string.h>  #include <vector>  #include <math.h>  #include <algorithm> | **Check Prime Number**:  bool **checkPrime(**long x**)** **{**  **if** **(**x **<=** 1**)** **return** **false;**  //if( x == 2 || x == 3) return true;  **if** **(**x **<=** 3**)** **return** **true;**  **if** **(!(**x **%** 2**)** **||** **!(**x **%** 3**))** **return** **false;**  **for** **(**long i **=** 5**;** i **\*** i **<=** x**;** i **+=** 6**)**  **if** **(!(**x**%**i**)** **||** **!(**x**%(**i **+** 2**)))** **return** **false;**  **return** **true;**  **}**  **Sieve of Eratosthenes**:  bitset<10000010> bs; vector<LL> primes;  SieveOfEratosthenes**(**ceil**(** sqrt**(**long long upper\_bound**)** **)** **)**  void **SieveOfEratosthenes(**long long upper\_bound**)** **{**  bs**.**set**();**  bs**[**0**]** **=** bs**[**1**]** **=** 0**;**  **for(**long long i **=** 2**;** i **<=** upper\_bound **+** 1**;** i**++)** **{**  **if(**bs**[**i**])** **{**  **for(**long long j **=** i **\*** i**;** j **<=** upper\_bound **+** 1**;** j **+=** i**)**  bs**[**j**]** **=** 0**;**  primes**.**push\_back**((**int**)** i**);**  **}**  **}**  **}**  **Sieve of Atkins:**  void **SieveOfAtkins**(LL n) {  vector<bool> isPrime(n + 1);  isPrime[2] = isPrime[3] = true;  for (LL i = 5; i <= n; i++) isPrime[i] = false;  for (LL x = 1; x \* x <= n; x++) {  for (LL y = 1; y \* y <= lim; y++) {  LL num = (4 \* x \* x + y \* y);  if (num <= n && (num%12==1 || num%12==5))  isPrime[num] = true;  num = (3 \* x \* x + y \* y);  if (num <= n && (num % 12 == 7))  isPrime[num] = true;  if (x > y) {  num = (3 \* x \* x - y \* y);  if (num <= n && (num % 12 == 11))  isPrime[num] = true;  } } }  for (LL i = 5; i <= lim; i++)  if (isPrime[i])  for (LL j = i \* i; j <= n; j += i)  isPrime[j]=false;  } |
| **Number of Divisors (Số các ước số của một số)**: divisors. Ex: 10 chia hết cho {1,2,5,10} = 4 divisors  **SieveOfEratosthenes**(1000000000);  long long **numberOfDivisors(**long long n**)** **{**  **if(**bs**[**n**]** **==** 1**)** **return** 2**;**  int ind **=** 0**;**  long long pf **=** primes**[**ind**];**  int result **=** 1**;**  **while** **(**pf **\*** pf **<=** n**)** **{**  int v **=** 0**;**  **for** **(;** n **%** pf **==** 0**;** v**++,** n **/=** pf**);**  pf **=** primes**[++**ind**];**  **if** **(**v **>** 0**)** result **\*=** **(**v **+** 1**);**  **}**  **if** **(**n **>** 1**)** result **\*=** 2**;**  **return** result**;**  **}**  **Tồng các ước số của một số:**  **SieveOfEratosthenes**(1000000000);  long long **sumOfDivisors(**long long n**)** **{**  **if(**bs**[**n**]** **==** 1**)** **return** n **+** 1**;**  int ind **=** 0**;**  long long pf **=** primes**[**ind**];**  long long result **=** 1**;**  **while** **(**pf **\*** pf **<=** n**)** **{**  int v **=** 0**;**  **for** **(;** n **%** pf **==** 0**;** v**++,** n **/=** pf**);**  **if** **(**v **>** 0**)**  result **\*=** **(**fastexp**(**pf**,**v**+**1**)-**1**)/(**pf**-**1**);**  pf **=** primes**[++**ind**];**  **}**  **if** **(**n **>** 1**)** result **\*=** **(**n**\***n **-** 1**)/(**n**-**1**);**  **return** result**;**  **}** | |
| **Fermat Little Test:** Let **a** be a random number between 2 and N-1, N has a **high probability** to be a prime number if N passes the test:  bool **FermatTest**(int a, int n) {  return (bigmod(a, n, n) == a); }  **Carmichael Numbers: non-prime** numbers that still pass the test. | | bool **isCarmichaelNumber**(int n) {  bool check = false;  int maxI = (int) ceil(pow(n, 1.0 / 3));  // Carmichael # has at least 3 prime factors  for (int i = 2; i <= maxI; i++) {  if (!**FermatTest**(i, n)) return false;  if (n % i == 0) check = true;  }  return check;  } |
| **Big Modulo**:  long **bigMod(**long b**,** long p**,** long m**)** **{**  **if** **(**p **==** 0**)** **return** 1**;**  **else** **if** **(**p **%** 2 **==** 0**)**  **return** square**(**bigMod**(**b**,** p **/** 2**,** m**))** **%** m**;**  **else**  **return** **((**b **%** m**)\*(**bigMod**(**b**,**p**-**1**,**m**)%**m**))%**m**;**  **}**  long **square**(long i) { return i\*i; } | | **Fast Exponential cách 1**:  long **exponent**(long a, long n) {  return exp(log(a) \* n);  }  **Cách 2:**  long long **fastexp**(long a, long n) {  if (n == 0) return 1;  else if (n%2==0) return square(fastexp(a,n/2));  else return a\*(fastexp(a,n-1));  } |
| **Greatest Common Divisor (Ước chung lớn nhất):**  int **GCD(**int x**,** int y**)** **{**  **if** **(!**x **||** **!**y**)** **return** x **>** y **?** x **:** y**;**  **for** **(**int i**;** i **=** x **%** y**;** x **=** y**,** y **=** i**);**  **return** y**;**  **}**  **GCD of 3 numbers**:  **Least Common Multiplier:** | | **Fibonacci**: f(n) = f(n-1) + f(n-2)  long **Fibonacci**(long n) {  int a = 1, b = 1, c = 0;  for (int i = 3; i <= n; a = b, b = c, ++i) {  c = a + b;  }  return a;  } |
| **Factorial**: n <= 20. Nếu n > 20 phải dùng ct số lớn  long long **Factorial** (long n) {  long long r = 1;  for (long i = 2; i <= n; ++i) {  r = r \* i;  }  return r;  } | | **Divisibility by 3**: if the sum of all digits % 3 == 0  bool **Divisibility3**(string str) {  int c;  long sum = 0;  for (int i = 0; i < str.size(); ++i) {  c = str.at(i) - '0';  sum = sum + c;  }  if (sum % 3 == 0) return true;  return false;  } |
| **Euler Totient Function**:  long **EulerTotient**(long n) {  long r = n;  for (int i = 2; i \* i <= n; ++i)  if (n % i == 0) {  while (n % i == 0) n /= i;  r -= r / i;  }  if (n > 1) r -= r / n;  return r;  } | | **Combination**: nCm  long **Combination**(long n, long k) {  if (k > n / 2) { k = n - k; }  long a, b, c = 1, d = 1;  for (long i = k; i > 0; --i) {  a = n - k + i;  b = i;  divideByGCD(a, b);  divideByGCD(c, b);  divideByGCD(a, d);  c = c \* a;  d = d \* b;  }  return (c / d);  } |
| void **divideByGCD**(long& a, long& b) {  long gcd = GCD(a, b);  a /= gcd; b /= gcd;  } | |
| **Extended GCD**: returns d=gcd(a,b), and give one pair x,y such that ax+by=d  long long Extended\_GCD**(**long long a**,** long long b**,**  long long **&**x**,** long long **&**y**)** **{**  long long quotient**,** temp**;**  long long tempX **=** 0**,** tempY **=** 1**;**  y **=** 0**;** x **=** 1**;**  **while** **(**b **>** 0**)** **{**  quotient **=** a **/** b**;**  temp **=** b**;**  b **=** a **%** b**;**  a **=** temp**;**  temp **=** tempX**;**  tempX **=** x **-** quotient**\***tempX**;**  x **=** temp**;**  temp **=** tempY**;**  tempY **=** y **-** quotient**\***tempY**;**  y **=** temp**;**  **}**  long long gcd **=** a**;**  **return** gcd**;**  **}** | | **Chinese Remainder Theorem**: Tìm số lượng N món hàng chưa biết. Chỉ biết nếu lấy , ,... Nhập vào 2 array và . Tính ra N.  long long modinverse**(**long long a**,** long long m**)** **{**  long long x**,** y**;**  Extended\_GCD**(**a **,**m **,**x **,**y**);**  **return** **(**x **%** m **+** m**)** **%** m**;**  **}**  long long chinese\_remainder**(**long long **\***modulo**,** long long **\***remainder**,** int size**)** **{**  **if** **(**size **==** 1**)** **return** **\***modulo**;**  long long tmp **=** modinverse**(**remainder**[**0**],**  remainder**[**1**]);**  long long tmp2 **=** **(**tmp **\*** **(**modulo**[**1**]** **–**  modulo**[**0**])** **%** remainder**[**1**]** **+** remainder**[**1**])**  **%** remainder**[**1**];**  long long ora **=** modulo**[**1**];**  long long tgcd **=** gcd**(**remainder**[**0**],**  remainder**[**1**]);**  modulo**[**1**]** **=** modulo**[**0**]** **+** remainder**[**0**]** **/** tgcd **\***  tmp2**;**  remainder**[**1**]** **\*=** remainder**[**0**]** **/** tgcd**;**  long long ret **=** chinese\_remainder**(**modulo **+** 1**,**  remainder **+** 1**,** size **-** 1**);**  remainder**[**1**]** **/=** remainder**[**0**]** **/** tgcd**;**  modulo**[**1**]** **=** ora**;**  **return** ret**;**  **}** |
| **Các Công Thức Khác:**   * Số lượng phép nhân trong matrix chain: A[n][m] x B[m][p] là * **Catalan Number**: cho n <= 100   long **CatalanNumber**(long n) {  return Combination(n \* 2, n) / (n + 1);  } |
| **Gambler’s Ruin**  **Unfair Coin Flipping: for problem in which 2 players toss the coin to win a turn with probabilities p and q = 1 – p respectively. The winner takes 1 point from other (player 1 has n1 socores, other is n2). The probability for each of them to win the game is:** | | **Frobenius Coin Problem:**  **Given positive integers a1, a2, ..., an such that gcd(a1, a2, ..., an) = 1, find the largest integer that cannot be expressed as an integer conical combination of these numbers, i.e., as a sum**  **k1a1 + k2a2 + ··· + knan.**   * **N=2, X = a1\*a2 – a1 – a2.** * **N > 2:**   int nums**[]** **=** **{**6**,** 9**,** 20**};**  int ns**[**100**];**  int m**;**  void frobenius**()**  **{**  ns**[**0**]** **=** 0**;**  memset**(**ns**,** **-**1**,** **sizeof** ns**);**  **for** **(**int i **=** 1**;** i **<** m**;** i**++)**  **{**  int d **=** gcd**(**nums**[**0**],** nums**[**i**]);**  **for** **(**int r **=** 0**;** r **<** d**;** r**++)**  **{**  int n **=** **-**1**;**  **if** **(**r **==** 0**)**  n **=** 0**;**  **else**  **{**  int q **=** r**;**  **while** **(**q **<** nums**[**0**])**  **{**  **if** **(**ns**[**q**]** **!=** **-**1 **&&** **(**ns**[**q**]** **<** n **||** n **==** **-**1**))**  n **=** ns**[**q**];**  q **+=** d**;**  **}**  **}**  **if** **(**n **!=** **-**1**)**  **for** **(**int j **=** 0**;** j **<** nums**[**0**]** **/** d**;** j**++)**  **{**  n **+=** nums**[**i**];**  int p **=** n **%** nums**[**0**];**  **if** **(**ns**[**p**]** **!=** **-**1 **&&** **(**ns**[**p**]** **<** n **||** n **==** **-**1**))**  n **=** ns**[**p**];**  ns**[**p**]** **=** n**;**  **}**  **}**  **}**  int max **=** 0**;**  **for** **(**int i **=** 0**;** i **<** nums**[**0**];** i**++)**  **if** **(**ns**[**i**]** **==** **-**1 **||** ns**[**i**]** **>** max**)**  max **=** ns**[**i**];**  **if** **(**max **==** **-**1**)** cout**<<** **-**1**;**  **else** cout**<<** max **-** nums**[**0**];**  **}** |

# Long Arithmetics (Phép tính số lớn)

|  |  |
| --- | --- |
| **Khai Báo Biến:**  **typedef** vector**<**int**>** lnum**;**  const int base **=** 1000 **\*** 1000 **\*** 1000**;**  lnum n**,** m**,** o**;**  **Input number**: input từ string s  void **Input(**string s**,** lnum**&** a**)** **{**  **for** **(**int i **=** **(**int**)**s**.**length**();** i**>**0**;** i**-=**9**)** **{**  **if** **(**i **<** 9**)**  a**.**push\_back**(**atoi**(**s**.**substr**(**0**,**i**).**c\_str**()));**  **else**  a**.**push\_back**(**atoi**(**s**.**substr**(**i**-**9**,** 9**).**c\_str**()));**  **}**  **while** **(**a**.**size**()** **>** 1 **&&** a**.**back**()** **==** 0**)**  a**.**pop\_back**();** //Remove leading 0s  **}** | **Print number**  void **Output(**lnum**&** a**)** **{**  printf**(**"%d"**,** a**.**empty**()** **?** 0 **:** a**.**back**());**  **for** **(**int i **=** **(**int**)** a**.**size**()** **-** 2**;** i **>=** 0**;** **--**i**)**  printf**(**"%09d"**,** a**[**i**]);**  **}**  **Compare:** return true if a >= b  bool **Compare(**lnum**&** a**,** lnum**&** b**)** **{**  **if** **(**a**.**size**()** **>** b**.**size**())** **return** **true;**  **else** **if** **(**a**.**size**()** **<** b**.**size**())** **return** **false;**  **else** **if** **(**a**.**back**()** **>=** b**.**back**())** **return** **true;**  **else** **return** **false;**  **}** |
| **Addition**: store result in a  void **Addition(**lnum**&** a**,** lnum**&** b**)** **{**  int carry **=** 0**;**  **for** **(**size\_t i **=** 0**;** i **<** max**(**a**.**size**(),**  b**.**size**())** **||** carry**;** **++**i**)** **{**  **if** **(**i **==** a**.**size**())** a**.**push\_back**(**0**);**  a**[**i**]** **+=** carry **+** **(**i **<** b**.**size**()** **?** b**[**i**]:**0**);**  carry **=** a**[**i**]** **>=** base**;**  **if** **(**carry**)** a**[**i**]** **-=** base**;**  **}**  **}** | **Subtraction**: a >= b. Store result in a  void **Subtraction(**lnum**&** a**,** lnum**&** b**)** **{**  int carry **=** 0**;**  **for** **(**size\_t i **=** 0**;** i**<**b**.**size**()||**carry**;** **++**i**)** **{**  a**[**i**]** **-=** carry **+** **(**i **<** b**.**size**()** **?** b**[**i**]:**0**);**  carry **=** a**[**i**]** **<** 0**;**  **if** **(**carry**)** a**[**i**]** **+=** base**;**  **}**  **while** **(**a**.**size**()** **>** 1 **&&** a**.**back**()** **==** 0**)**  a**.**pop\_back**();**  **}** |
| **Multiply 2 số lớn**: multiply a and b, store result in c  void **Multiply(**lnum**&** a**,** lnum**&** b**,** lnum **&**c**)** **{**  c**.**resize**(**a**.**size**()** **+** b**.**size**());**  **for** **(**size\_t i **=** 0**;** i **<** a**.**size**();** **++**i**)** **{**  **for** **(**int j **=** 0**,** carry **=** 0**;** j **<** **(**int**)** b**.**size**()** **||** carry**;** **++**j**)** **{**  long long cur **=** c**[**i **+** j**]** **+** a**[**i**]** **\*** 1ll **\*** **(**j **<** **(**int**)** b**.**size**()** **?** b**[**j**]** **:** 0**)** **+** carry**;**  c**[**i **+** j**]** **=** **(**int**)** cur **%** base**;**  carry **=** **(**int**)** cur **/** base**;**  **}**  **}**  **while(**c**.**size**()** **>** 1 **&&** c**.**back**()** **==** 0**)** c**.**pop\_back**();**  **}** | |
| **Multiply 1 số lớn vs 1 số bé**:  void **MultiplyShort(**lnum**&** a**,** int b**)** **{**  long long carry **=** 0**;**  **for** **(**int i **=** 0**;** i **<** a**.**size**()** **||** carry**;** **++**i**)** **{**  **if** **(**i **==** a**.**size**())** **{**  a**.**push\_back**(**0**);**  **}**  long long cur **=** a**[**i**]** **\*** 1ll **\*** b **+** carry**;**  a**[**i**]** **=** **(**int**)** **(**cur **%** base**);**  carry **=** **(**long long**)** **(**cur **/** base**);**  **}**  **while** **(**a**.**size**()** **>** 1 **&&** a**.**back**()** **==** 0**)**  a**.**pop\_back**();**  **}** | **Divide 1 số lớn vs 1 số bé**: a/b (b là 1 số long < 1000000000), store result in a  void **DivideShort(**lnum**&** a**,** int b**)** **{**  int carry **=** 0**;**  **for** **(**int i **=** **(**int**)** a**.**size**()** **-** 1**;** i**>=**0**;** **--**i**)** **{**  long long cur **=** a**[**i**]+**carry**\***1ll**\***base**;**  a**[**i**]** **=** **(**int**)** **(**cur **/** b**);**  carry **=** **(**int**)** **(**cur **%** b**);**  **}**  **while** **(**a**.**size**()** **>** 1 **&&** a**.**back**()** **==** 0**)**  a**.**pop\_back**();**  **}** |
| **Large Fibonacci:**  void **LargeFibonacci**(int n) {  lnum a, b, c;  a.push\_back(1);  b.push\_back(1);  for (int i = 3; i <= n; ++i) {  c = a;  Add(a, b);  b = c;  }  Output(a);  } | **Large Catalan:**  void **LargeCatalan**(int n) {  lnum a;  a.push\_back(1);  for (int i = n + 2; i <= n \* 2; ++i) {  MultiplyShort(a, i);  }  for (int i = 1; i <= n; ++i) {  DivideShort(a, i);  }  Output(a);  } |

# Search & Sort

|  |  |
| --- | --- |
| **Binary Search:**  int **binary\_search**(int A[], int len, int key) {  int l = 1, r = len;  int m;  while (r - l > 1) {  m = l + (r - l) / 2;  if (A[m] >= key) r = m;  else l = m;  }  return r;  } | **Quick Sort:**  int A[10001];  void **quicksort**(int l, int r) {  if (l >= r) return;  int i = l;  int j = r;  int x = A[(l + r) / 2]; //x=A[random(l-r)]  while (i <= j) {  while (A[i] < x) i++;  while (A[j] > x) j--;  if (i <= j) {  swap(A[i], A[j]);  i++;  j--;  }  }  quicksort(l, j);  quicksort(i, r);  } |

# Dynamic Programming

## Bad Neighbors (Bài toán DP theo dạng hình tròn).

//1 khu phố có nhà xếp theo 1 hình tròn. 1 nhà chỉ donate money khi 2 nhà 2 bên không donate money và ngược lại. Tìm cách thu số money lớn nhất?

vector<int> n;

int **collectMoney()** **{**

//Special case number of element <= 2

**if** **(**n**.**size**()** **==** 1**)** **{**

cout **<<** "Result = " **<<** n**[**0**]** **<<** "\n"**;**

**}** **else** **if** **(**n**.**size**()** **==** 2**)** **{**

cout**<<**"Result = "**<<**max**(**n**[**0**],** n**[**1**])<<**"\n"**;**

**}**

//Initialize 2 array - 1st array remove last neighbor - 2nd array remove 1st neighbor

vector**<**int**>** n1**(**n**.**begin**(),** n**.**end**()** **-** 1**);**

vector**<**int**>** n2**(**n**.**begin**()** **+** 1**,** n**.**end**());**

//Formula: n[i] += max(n[i-2], n[i-3])

**for** **(**int i **=** 2**;** i **<** n1**.**size**();** i**++)** **{**

**if** **(**i **>=** 3**)** **{**

n1**[**i**]** **+=** max**(**n1**[**i **-** 2**],** n1**[**i **-** 3**]);**

n2**[**i**]** **+=** max**(**n2**[**i **-** 2**],** n2**[**i **-** 3**]);**

**}** **else** **{**

n1**[**i**]** **+=** n1**[**i **-** 2**];**

n2**[**i**]** **+=** n2**[**i **-** 2**];**

**}**

**}**

**return** max**(**n1**.**back**(),** n2**.**back**());;**

**}**

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## Matrix Chain Multiplication (Stripe Painter)

string s;

int **stripePainter**() {

int m[s.length()][s.length()];

int i, j, k;

for (i = 0; i < s.length(); i++) {

for (j = 0; j < s.length(); j++) {

if (i == j) m[i][i] = 1;

else m[i][j] = MAX;

}

}

for (i = 1; i < s.length(); i++) {

for (j = 0; j < s.length() - i; j++) {

for (k = j; k < j + i; k++) {

m[j][j+i] = min(m[j][j+i], m[j][k]

+ m[k+1][j+i] - (s.at(j)

== s.at(j+i) ? 1:0));

}

}

}

return m[0][s.length() - 1];

}

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## Longest Common Subsequence

|  |  |
| --- | --- |
| Cách 1: O(nm) nhưng tốn space  int m[10000][10000];  int t[10000][10000];  string s1, s2;  int **LCS**() {  int i, j, M, N;  M = s1.length();  N = s2.length();  for (i = M; i >= 0; i--) {  for (j = N; j >= 0; j--) {  if (i == M || j == N) {  m[i][j] = 0;  t[i][j] = 0;  continue;  }  if (s1[i] == s2[j]) {  m[i][j] = 1 + m[i + 1][j + 1];  t[i][j] = 1;  } else {  m[i][j] = max(m[i][j+1],m[i+1][j]);  t[i][j]=(m[i][j+1] > m[i+1][j]) ? 2:3;  }  }  }  return m[0][0];  } | Cách 2: O(nm) tốn O(n) space  int m[2][10000];  string s1, s2;  int **LCS**() {  int i, j, M, N, ii;  M = s1.length();  N = s2.length();    for (i = M; i >= 0; i--) {  ii = i & 1;  for (j = N; j >= 0; j--) {  if (i == M || j == N) {  m[ii][j] = 0;  continue;  }  if (s1[i] == s2[j]) {  m[ii][j] = 1 + m[1 - ii][j + 1];  }  else {  m[ii][j] = max(m[ii][j+1], m[1-ii][j]);  }  }  }  return m[0][0];  } |

//Gọi trace: Trace\_Back(0,0,s1.length(), s2.length())

void **Trace\_Back**(int i, int j, int M, int N) {

if (i == M || j == N) {

return;

} else if (t[i][j] == 1) {

cout << s1[i] << **"** **"**;

Trace\_Back(i + 1, j + 1);

} else if (t[i][j] == 3) {

Trace\_Back(i + 1, j);

} else {

Trace\_Back(i, j + 1);

}

}

------------------------------------------------------------------------------------------------------

## Longest Common Palindrome Subsequence

int matrix**[**MAX\_ITEM**][**MAX\_ITEM**];**

int trace**[**MAX\_ITEM**][**MAX\_ITEM**];**

ostringstream result**;**

string str**;**

void **Longest\_Palindrome\_String()** **{**

int i**,** j**,** k**;**

**for** **(**i **=** 0**;** i **<** str**.**length**();** **++**i**)** **{**

**for** **(**int j **=** 0**;** j **<** str**.**length**();** **++**j**)** **{**

matrix**[**i**][**j**]** **=** 0**;**

trace**[**i**][**j**]** **=** 0**;**

**}**

matrix**[**i**][**i**]** **=** 1**;**

**}**

**for** **(**k **=** 2**;** k **<=** str**.**length**();** **++**k**)** **{**

**for** **(**i **=** 0**;** i **<** str**.**length**()** **-** k **+** 1**;** **++**i**)** **{**

j **=** i **+** k **-** 1**;**

**if** **(**str**[**i**]** **==** str**[**j**]** **&&** k **==** 2**)** **{**

matrix**[**i**][**j**]** **=** 2**;**

**}** **else** **if** **(**str**[**i**]** **==** str**[**j**])** **{**

matrix**[**i**][**j**]** **=** matrix**[**i**+**1**][**j**-**1**]** **+** 2**;**

**}** **else** **{**

**if(**matrix**[**i**+**1**][**j**]** **>=** matrix**[**i**][**j**-**1**]){**

matrix**[**i**][**j**]** **=** matrix**[**i **+** 1**][**j**];**

trace**[**i**][**j**]** **=** 1**;**

} **else** {

matrix**[**i**][**j**]** **=** matrix**[**i**][**j **-** 1**];**

trace**[**i**][**j**]** **=** **-**1**;**

}

}

}

}

}

void Trace\_Back**(**int l**,** int h**)** **{**

**if** **(**l **>** h**)** **return;**

**if** **(**l **==** h**)** **{**

result **<<** str**[**l**];**

**return;**

**}**

**if** **(**trace**[**l**][**h**]** **==** 0**)** **{**

result **<<** str**[**l**];**

Trace\_Back**(**l **+** 1**,** h **-** 1**);**

result **<<** str**[**h**];**

**}** **else** **if** **(**trace**[**l**][**h**]** **==** 1**)**

Trace\_Back**(**l **+** 1**,** h**);**

**else**

Trace\_Back**(**l**,** h **-** 1**);**

**}**

int **main**(int argc, char\*\* argv) {

cin >> str;

Longest\_Palindrome\_String();

Trace\_Back(0, str.length() - 1);

cout << result.str() << endl;

return 0;

}

------------------------------------------------------------------------------------------------------

## Longest Increasing Subsequence

|  |  |
| --- | --- |
| **O(nlogn) – dùng set**  int a**[**30000**];**  set**<**int**>** st**;**  set**<**int**>::**iterator it**;**  int **main** **()** **{**  // Input  int N**,** t**;**  cin **>>** N**;**  **for** **(**int i **=** 0**;** i **<** N**;** **++**i**)** **{**  cin **>>** t**;**  a**[**i**]** **=** t**;**  **}**  //Thuat toan  st**.**clear**();**  **for(**int i**=**0**;** i**<**N**;** i**++)** **{**  it **=** st**.**lower\_bound**(**a**[**i**]);**  **if** **(**it **!=** st**.**end**())** st**.**erase**(**it**);**  st**.**insert**(**a**[**i**]);**  **}**  //In ket qua  cout**<<**st**.**size**()<<**endl**;**  **for** **(**it **=** st**.**begin**();** it **!=** st**.**end**();** **++**it**)** **{**  cout **<<** **\***it**;**  **}**  cout **<<** endl**;**  **return** 0**;**  **}** | **O(nlogn) – dùng binary search**  int m**[**2**][**10000**];**  string s1**,** s2**;**  int **LCS()** **{**  int i**,** j**,** M**,** N**,** ii**;**  M **=** s1**.**length**();**  N **=** s2**.**length**();**    **for** **(**i **=** M**;** i **>=** 0**;** i**--)** **{**  ii **=** i **&** 1**;**  **for** **(**j **=** N**;** j **>=** 0**;** j**--)** **{**  **if** **(**i **==** M **||** j **==** N**)** **{**  m**[**ii**][**j**]** **=** 0**;**  **continue;**  **}**  **if** **(**s1**[**i**]** **==** s2**[**j**])** **{**  m**[**ii**][**j**]** **=** 1 **+** m**[**1 **-** ii**][**j **+** 1**];**  **}**  **else** **{**  m**[**ii**][**j**]** **=** max**(**m**[**ii**][**j**+**1**],** m**[**1**-**ii**][**j**]);**  **}**  **}**  **}**  **return** m**[**0**][**0**];**  **}** |

------------------------------------------------------------------------------------------------------

## Levenshtein Distance

//Tìm số lần edit, swap, add, delete chuỗi string 1 sao cho ra chuỗi string 2

string A**,** B**;**

int matrix**[**MAX\_ITEM**][**MAX\_ITEM**];**

int **Levenshtein()** **{**

int i**,** j**;**

**for** **(**i **=** 1**;** i **<=** A**.**length**();** **++**i**)** matrix**[**i**][**0**]** **=** i**;**

**for** **(**i **=** 1**;** i **<=** B**.**length**();** **++**i**)** matrix**[**0**][**i**]** **=** i**;**

**for** **(**i **=** 1**;** i **<=** A**.**length**();** **++**i**)** **{**

**for** **(**j **=** 1**;** j **<=** B**.**length**();** **++**j**)** **{**

matrix**[**i**][**j**]** **=** min**(**min**(**matrix**[**i **-** 1**][**j**]** **+** 1**,** matrix**[**i**][**j **-** 1**]** **+** 1**),**

matrix**[**i **-** 1**][**j **-** 1**]** **+** **((**A**[**i **-** 1**]** **==** B**[**j **-** 1**])** **?** 0 **:** 1**));**

**}**

**}**

**return** matrix**[**i **-** 1**][**j **-** 1**];**

**}**

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## Knapsack Problem

|  |  |
| --- | --- |
| **Classic Knapsack Problem** (Unbounded knapsack problem)  // Có Nmón hàng. Mỗi món hàng có w khối lượng và v giá trị. Có 1 túi xách đựng được tối đa **W** khối lượng. Tìm cách lấy nhiều món hàng nhất có thể với tổng giá trị lớn nhất.  int w**[**100**];** // weight  int v**[**100**];** // value  int m**[**100**][**10000**];** // dynacmic table arr  int t**[**100**][**10000**];** // trace array  void **Trace\_Back(**int i**,** int j**)** **{**  **if** **(**i **==** 0 **||** j **==** 0**)** **return;**  **else** **if** **(**t**[**i**][**j**]** **==** 1**)** **{**  Trace\_Back**(**i **-** 1**,** j **-** w**[**i**]);**  result **<<** i **<<** " "**;**  **}** **else** **{**  Trace\_Back**(**i **-** 1**,** j**);**  **}**  **}**  //Classic Knapsack Problem (Unbounded knapsack problem)  // Có N món hàng. Mỗi món hàng có w khối lượng và v giá trị.  // Có 1 túi xách đựng được tối đa W khối lượng.  // Tìm cách lấy nhiều món hàng nhất có thể với tổng giá trị lớn nhất.  void **Knapsack\_Algorithm(**int N**,** int W**)** **{**  int i**,** j**;**  **for** **(**i **=** 1**;** i **<=** W**;** **++**i**)** m**[**0**][**i**]** **=** t**[**0**][**i**]** **=** 0**;**  **for** **(**i **=** 1**;** i **<=** N**;** **++**i**)** **{**  **for** **(**j **=** 1**;** j **<=** W**;** **++**j**)** **{**  **if** **(**j **>=** w**[**i**])** **{**  m**[**i**][**j**]** **=** max**(**m**[**i**-**1**][**j**],**  m**[**i**-**1**][**j**-**w**[**i**]]** **+** v**[**i**]);**  t**[**i**][**j**]** **=** **(**m**[**i**][**j**]==**m**[**i**-**1**][**j**])** **?** 0**:**1**;**  **}** **else** **{**  m**[**i**][**j**]** **=** m**[**i **-** 1**][**j**];**  t**[**i**][**j**]** **=** 0**;**  **}**  **}**  **}**  **Trace\_Back(**N**,** W**);**  **}** | **Exactly V value with minimum item**  // Có N tờ tiền, mỗi tờ tiền có v giá trị.  // Tìm cách lấy số tờ tiền ít nhất với tổng giá trị = V cho trước.  // Note: Bài này dùng Trace\_Back riêng để xuất ra giá trị của biến được chọn  void **Trace\_Back(**int i**,** int j**)** **{**  **if** **(**i **==** 0 **||** j **==** 0**)** **return;**  **else** **if** **(**t**[**i**][**j**]** **==** 1**)** **{**  Trace\_Back**(**i **-** 1**,** j **-** v**[**i**]);**  cout **<<** v**[**i**]** **<<** " "**;**  **}** **else** **{**  Trace\_Back**(**i **-** 1**,** j**);**  **}**  **}**  void **Knapsack\_Algorithm(**int N**,** int V**)** **{**  int i**,** j**,** temp **=** 0**;**  sort**(**v**,** v**+**N**);**  **for** **(**i **=** 1**;** i **<=** N**;** **++**i**)** temp **+=** v**[**i**];**  **if** **(**temp **<** V**)** **{**  cout **<<** "No Solution"**;** **return;**  **}**  **for** **(**i **=** 1**;** i **<=** V**;** **++**i**)** m**[**0**][**i**]** **=**t**[**0**][**i**]=** 0**;**  **for** **(**i **=** 1**;** i **<=** N**;** **++**i**)** **{**  **for** **(**j **=** 1**;** j **<=** V**;** **++**j**)** **{**  **if** **(**j **>=** v**[**i**])** **{**  m**[**i**][**j**]** **=** max**(**m**[**i**-**1**][**j**],**  m**[**i**-**1**][**j**-**v**[**i**]]** **+** v**[**i**]);**  t**[**i**][**j**]** **=** m**[**i**-**1**][**j**]** **==** m**[**i**][**j**]** **?** 0 **:** 1 **;**  **if(**m**[**i**][**j**]** **==** j**)** t**[**i**][**j**]** **=** 1**;**  **}** **else** m**[**i**][**j**]** **=** m**[**i**-**1**][**j**];**  **}**  **}**  **if** **(**m**[**N**][**V**]** **==** V**)** Trace\_Back**(**N**,** V**);**  **else** cout **<<** "No Solution"**;**  **}**  **Largest weight of all items that <= W**  // Có N cục đá, mỗi cục có w khối lượng.  // Tìm cách lấy sao cho số lượng đá là nhiều nhất.  int Trace\_Back**(**int i**,** int j**,** int count**)** **{**  **if** **(**i **==** 0 **||** j **==** 0**)** **return** count**;**  **else** **if** **(**t**[**i**][**j**]** **==** 1**)** **{**  count **++;**  Trace\_Back**(**i **-** 1**,** j **-** v**[**i**],** count**);**  //cout << v[i] << " ";  **}** **else** **{**  Trace\_Back**(**i **-** 1**,** j**,** count**);**  **}**  **}**  int Knapsack\_Algorithm**(**int N**,** int W**)** **{**  int i**,** j**,** temp **=** 0**;**  **for** **(**i **=** 1**;** i **<=** N**;** **++**i**)** temp **+=** w**[**i**];**  **if** **(**temp **<** W**)** **{**  **return** 0**;**  **}**  **for** **(**i **=** 1**;** i **<=** W**;** **++**i**)** m**[**0**][**i**]** **=**t**[**0**][**i**]=** 0**;**  **for** **(**i **=** 1**;** i **<=** N**;** **++**i**)** **{**  **for** **(**j **=** 1**;** j **<=** W**;** **++**j**)** **{**  **if** **(**j **>=** w**[**i**])** **{**  m**[**i**][**j**]** **=** max**(**m**[**i**-**1**][**j**],**  m**[**i**-**1**][**j**-**w**[**i**]]** **+** w**[**i**]);**  t**[**i**][**j**]** **=** m**[**i**-**1**][**j**]** **==** m**[**i**][**j**]** **?** 0 **:** 1 **;**  **}** **else** m**[**i**][**j**]** **=** m**[**i**-**1**][**j**];**  **}**  **}**  **if** **(**m**[**N**][**W**]** **==** W**)** **{** **return** Trace\_Back**(**N**,** W**,** 0**);** **}**  **else** **return** 0**;**  **}** |
| **Exactly W weight**  // kiểm tra xem có thể bỏ chính xác maxWeight khối lượng cục đá vào túi ko?  // W ở đây là total weight của N cục đá  int **Knapsack\_Algorithm(**int N**,** int W**)** **{**  int i**,** j**;**  **for** **(**i **=** 1**;** i **<=** W**;** **++**i**)** m**[**0**][**i**]** **=** INF**;**  **for** **(**i **=** 1**;** i **<=** N**;** **++**i**)** **{**  **for** **(**j **=** 1**;** j **<=** W**;** **++**j**)** **{**  **if** **(**j **>=** w**[**i**])** **{**  m**[**i**][**j**]** **=** min**(**m**[**i**-**1**][**j**],**  m**[**i**-**1**][**j**-**w**[**i**]]** **+** 1**);**  **}** **else** **{**  m**[**i**][**j**]** **=** m**[**i **-** 1**][**j**];**  **}**  **}**  **}**  **if** **(**m**[**N**][**maxWeight**]** **==** INF**)** **return** 0**;**  **return** m**[**N**][**maxWeight**];**  **}** |

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## Subset Sum Problem (TH đặc biệt của Knapsack)

|  |  |
| --- | --- |
| **Classic Subset Sum**  //Cho 1 chuỗi n số. Kiểm tra xem có subset nào trong chuỗi này có tổng bằng S cho trước ko  int a**[**1000**];** // store value of set  int m**[**1000**];** // store number of subset  //N is number of elements  //M is maximum value the set can get  //S is the specific value for counting subset  int SubsetSum**(**int N**,** int M**,** int S**)** **{**  int i**,** j**;**  **for** **(**i **=** 0**;** i **<** M **+** 10**;** i**++)** m**[**i**]** **=** 0**;**  m**[**0**]** **=** 1**;**  **for** **(**i **=** 0**;** i **<** N**;** i**++)**  **for** **(**j **=** M**;** j **>=** a**[**i**];** j**--)**  m**[**j**]** **+=** m**[**j **-** a**[**i**]];**  **}**  int main**()** **{**  int N**,** t**,** M **=** 0**,** S **=** 0**;**  cin **>>** N**;**  **for** **(**int i **=** 0**;** i **<** N**;** **++**i**)** **{**  cin **>>** t**;**  a**[**i**]** **=** t**;**  M **+=** t**;**  **}**  cin **>>** S**;**  SubsetSum**(**N**,** M**,** S**);**    **for** **(**int i **=** 0**;** i **<** M**;** **++**i**)** **{**  cout **<<** m**[**i**]** **<<** " "**;**  **}**  cout **<<** endl**;**  cout **<<** m**[**S**]** **<<** endl**;**  **return** 0**;**  **}** | **Các dạng biến thể:**   * **Candy for two kids**: We want to divide the candies as evenly as possible between the two kids. Now the problem is not to hit a fixedK.We want to search a K such that it is as close to M/2 as possible. We may simple compute the m array, and look up which is the nearest ”yes” below M/2. * **Subset sum with multiple supplies**: Each ai can be used as many times as you like in the sum, are we going to hit K? The solution is just by reversing the direction of the j-loop in the subset sum program: for(j=a[i];j<=K;j++) * **Coin counting**: có n đồng xu với mệnh giá khác nhau. Có bao nhiêu cách chọn 1 subset xu sao cho có tổng giá = K. The solution:   for(j = a[i]; j <= K; ++j) {  m[j] += m[j - a[i]];  } |

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## Partition (chia làm 2 phần bằng nhau)

string **Find\_Partition**(int N) {

int i, j, sum = 0;

sort(w, w + N + 1);

for (i = 1; i <= N; i++) sum += w[i];

if (sum % 2 != 0) return **"**No Solution**"**;

sum /= 2;

for (i = 0; i <= N; i++) m[i][0] = MAX\_VALUE;

for (j = 1; j <= sum; j++) m[0][j] = MAX\_VALUE;

m[0][0] = 0;

for (i = 1; i <= N; i++) {

for (j = 1; j <= sum; j++) {

m[i][j] = m[i - 1][j];

if (w[i] <= j) {

m[i][j] = min(m[i - 1][j], m[i - 1][j - w[i]] + 1);

t[i][j] = (m[i - 1][j] > m[i - 1][j - w[i]] + 1) ? 1 : 0;

}

}

}

if (m[N][sum] == N / 2) return Trace\_Back(N, sum);

return **"**No Solution**"**;

}

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## DIV

#define SIZE 200

#define MAX 10000

int Index[SIZE], K1[SIZE], K2[SIZE];

int Number[MAX], D[MAX];

int **main**() {

int length, numberOfDivisor, temp;

for (d = 0; d < numberOfDivisor; d++) {

fill(K1, K1 + 2 \* D[d], 0);

K1[Number[0] % D[d] + D[d]] = 1;

for (i = 1; i < length; i++) {

fill(K2, K2 + 2 \* D[d], 0);

for (j = 1; j < 2 \* D[d]; j++) {

if (K1[j]) {

K2[(j - D[d] + Number[i]) % D[d] + D[d]] = 1;

K2[(j - D[d] - Number[i]) % D[d] + D[d]] = 1;

}

}

copy(K2, K2 + 2 \* D[d], K1);

}

cout << K1[D[d]] << **"** **"**;

}

return 0;

}

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## Tick

#define MAX 1000000

#define SIZE 100000

int matrix[SIZE][3];

int **main**() {

int length;

matrix[0][2] = 0;

matrix[1][2] = matrix[1][0];

for (int i = 2; i <= length; i++) {

matrix[i][2] = min(matrix[i - 1][2] + matrix[i][0], matrix[i - 2][2] + matrix[i - 1][1]);

}

cout << matrix[length][2] << endl;

return 0;

}

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## RTicket

#define SIZE 1000000

#define MAX 1000000000

long long D[SIZE], minCost[SIZE];

int **main**() {

long long length, start, end, L1, L2, L3, C1, C2, C3, distance;

input >> L1 >> L2 >> L3 >> C1 >> C2 >> C3 >> length >> start >> end;

for (int i = 1; i < length; i++) in >> D[i];

D[0] = 0;

if (--start > --end) swap(start, end);

minCost[start] = 0;

for (int i = start + 1; i <= end; i++) {

minCost[i] = MAX;

for (int j = i - 1; j >= start && (distance = D[i] - D[j]) <= L3; j--) {

if (distance <= L1) minCost[i] = min(minCost[i], minCost[j] + C1);

else if (distance <= L2) minCost[i] = min(minCost[i], minCost[j] + C2);

else minCost[i] = min(minCost[i], minCost[j] + C3);

}

}

cout << minCost[end] << endl;

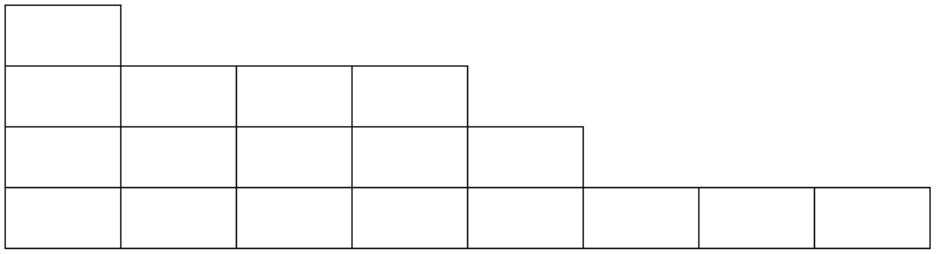
return 0;

}

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## Ladder

//Tìm số cách xếp N cube thành ladder. Hàng dưới phải nhiều hơn hàng trên ít nhất 1 cube



int matrix[N][N]

int ladder(int N) {

int i, j;

for (i = 0; i < N + 1; i++) {

matrix[0][i] = 0;

matrix[1][i] = 1;

}

for (i = 0; i < N + 1; i++) {

matrix[i][0] = 0;

}

for (i = 1; i <= N; i++) {

for (j = 1; j <= N; j++) {

if (i == 1) break;

else if (i == j) matrix[i][j] = matrix[i][j - 1] + 1;

else if (j > i) matrix[i][j] = matrix[i][j - 1];

else matrix[i][j] = matrix[i][j - 1] + matrix[i - j][j - 1];

}

}

return matrix[N][N];

}

## Domino

// Cho K domino co dang 2x1, sap xep K dominos vao matrix kich thuoc Nx3 sao cho tong weight tren cac dominos la cao nhat. Dung bitmask

#include <cstdio>

#include <cassert>

#include <algorithm>

**typedef** long long llint**;**

**typedef** int mask\_t**;**

const llint oo **=** 0x3f3f3f3f**;**

const int maxn **=** 1001**;**

const int maxk **=** 1001**;**

const int NS **=** 11**;**

const mask\_t slucajevi**[**NS**]** **=** **{**0x00**,** 0x11**,** 0x22**,** 0x44**,** 0x03**,**

0x06**,** 0x17**,** 0x47**,** 0x33**,** 0x55**,** 0x66**};** // co 11 cach sap xep dimino

const int slucajevi\_cost**[**NS**]** **=** **{**0**,** 1**,** 1**,** 1**,** 1**,** 1**,** 2**,** 2**,** 2**,** 2**,** 2**};**

int N**,** K**;**

int map**[**3**][**maxn**];**

llint dp**[**2**][**8**][**maxk**];**

void load**()** **{**

freopen**(**"inp.txt"**,** "r"**,** stdin**);**

scanf**(**"%d%d"**,** **&**N**,** **&**K**);**

**for** **(**int j **=** 0**;** j **<** N**;** **++**j**)**

**for** **(**int i **=** 0**;** i **<** 3**;** **++**i**)**

scanf**(**"%d"**,** **&**map**[**i**][**j**]);**

**}**

llint get**(**mask\_t slucaj**,** int n**)** **{**

llint ret **=** 0**;**

**for** **(**int i **=** 0**;** i **<** 2**;** **++**i**,** slucaj **>>=** 4**)** **{**

**if** **(**n **-** i **<** 0 **&&** slucaj**)** **{**

**return** **-**oo**;**

**}**

**for** **(**int j **=** 0**;** j **<** 3**;** **++**j**)** **{**

ret **+=** **(**slucaj **&** **(**1 **<<** j**))** **?** map**[**j**][**n **-** i**]** **:** 0**;**

**}**

**}**

**return** ret**;**

**}**

llint solve**()** **{**

int read **=** 0**;**

**for** **(**int k **=** 1**;** k **<=** K**;** **++**k**)** **{**

**for** **(**mask\_t right **=** 0**;** right **<** 8**;** **++**right**)** **{**

dp**[**read**][**right**][**k**]** **=** **-**oo**;**

**}**

**}**

**for** **(**int i **=** 0**;** i **<** N**;** **++**i**)** **{**

**for** **(**int k **=** 0**;** k **<=** K**;** **++**k**)** **{**

// init

**for** **(**mask\_t right **=** 0**;** right **<** 8**;** **++**right**)** **{**

dp**[!**read**][**right**][**k**]** **=** **-**oo**;**

**}**

**for** **(**int ns **=** 0**;** ns **<** NS**;** **++**ns**)** **{**

int cost **=** slucajevi\_cost**[**ns**];**

mask\_t slucaj **=** slucajevi**[**ns**];**

mask\_t left\_busy **=** **(**slucaj **>>** 4**)** **&** 0xf**;**

mask\_t right\_busy **=** slucaj **&** 0xf**;**

**if** **(**k **-** cost **<** 0**)** **break;**

**for** **(**mask\_t right **=** 0**;** right **<** 8**;** **++**right**)** **{**

**if** **(**right **&** right\_busy**)** **continue;**

llint **&**ref **=** dp**[!**read**][**right**][**k**];**

ref **=** std**::**max**(**ref**,** dp**[**read**][**left\_busy**][**k **-** cost**]** **+** get**(**slucaj**,** i**));**

**}**

**}**

**}**

read **=** **!**read**;**

**}**

**return** dp**[**read**][**0**][**K**];**

**}**

int main**()** **{**

load**();**

printf**(**"%lld\n"**,** solve**());**

**return** 0**;**

**}**

## Winning streak

//Cho n trận đấu và xác suất chiến thắng p của đội A. Tìm giá trị lớn nhất của chuỗi trận thắng liên tiếp của đội A sau n trận đấu. The expected value of the longest streak is the average of the longest in all possible outcomes of all games in a season, weighted by their probability. For instance, assume that the season consists of only three games, and that p = 0.4. In this case, the expected length of the longest winning streak becomes 0.216·0 + 0.144·1 + 0.144·1 + 0.096·2 + 0.144·1 + 0.096·1 + 0.096·2 + 0.064·3 = 1.104 (LLL + LLW + LWL + WLL + LWW + WLW + WWL + WWW)

// dp[i][j] will have the probability of having a maximum winning streak of

// j in i consecutive games (tức là xác suất thắng liên tiếp dc 1, 2, ... j trận liên tiếp sau i trận)

double dp[MAXN + 1][MAXN + 1];

// pp[i] will have p^i

double pp[MAXN + 1];

int n;       // from the input, the number of consecutive games

double p;    // from the input, the probability of winning 1 game

double ans;

void run\_dp()

{

    pp[0] = 1.0;

    for (int i = 1; i <= n; ++i) pp[i] = pp[i - 1] \* p;

    for (int j = 0; j <= n; ++j)

        dp[0][j] = 1.0;

    for (int i = 1; i <= n; ++i)

        for (int j = 0; j <= n; ++j) {

            dp[i][j] = dp[i-1][j];

            if (j == i - 1)

                dp[i][j] -= pp[i];

            else if (j < i - 1)

                dp[i][j] -= dp[i - j - 2][j] \* (1-p) \* pp[j+1];

        }

    ans = 0.0;

    for (int j = 1; j <= n; ++j)

        ans += j \* (dp[n][j] - dp[n][j - 1]);

}

// To get our answer we just need to iterate over the nth row of our data structure, and use the fact that f(n,j)−f(n,j−1) is the probability of having a winning streak of exactly j in n games.

// Tức là ta tính sum ( j \* ( f(n,j)−f(n,j−1) ) ) với j chạy từ 1->n

# Dates

*// Routines for performing computations on dates. In these routine months are expressed as integers from 1 to 12, days are expressed as integers from 1 to 31, and years are expressed as 4-digit integers.*

|  |  |
| --- | --- |
| string dayOfWeek**[]** **=** **{**"Mon"**,** "Tue"**,** "Wed"**,** "Thu"**,** "Fri"**,** "Sat"**,** "Sun"**};**  // converts Gregorian date to integer (Julian day number)  int dateToInt**(**int m**,** int d**,** int y**)** **{**  **return**  1461 **\*** **(**y **+** 4800 **+** **(**m **-** 14**)** **/** 12**)** **/** 4 **+**  367 **\*** **(**m **-** 2 **-** **(**m **-** 14**)** **/** 12 **\*** 12**)** **/** 12  **-** 3**\*((**y **+** 4900 **+** **(**m **-** 14**)** **/** 12**)/**100**)/**4 **+**  d **-** 32075**;**  **}**  // converts integer (Julian day number) to day of week  string intToDay**(**int jd**)** **{**  **return** dayOfWeek**[**jd **%** 7**];**  **}** | // converts integer (Julian day number) to Gregorian date: month/day/year  void intToDate**(**int jd**,** int **&**m**,** int **&**d**,** int **&**y**)** **{**  int x**,** n**,** i**,** j**;**  x **=** jd **+** 68569**;**  n **=** 4 **\*** x **/** 146097**;**  x **-=** **(**146097 **\*** n **+** 3**)** **/** 4**;**  i **=** **(**4000 **\*** **(**x **+** 1**))** **/** 1461001**;**  x **-=** 1461 **\*** i **/** 4 **-** 31**;**  j **=** 80 **\*** x **/** 2447**;**  d **=** x **-** 2447 **\*** j **/** 80**;**  x **=** j **/** 11**;**  m **=** j **+** 2 **-** 12 **\*** x**;**  y **=** 100 **\*** **(**n **-** 49**)** **+** i **+** x**;**  **}** |
| int main**(**int argc**,** char **\*\***argv**)** **{**  int jd **=** dateToInt**(**3**,** 24**,** 2004**);**  int m**,** d**,** y**;**  intToDate**(**jd**,** m**,** d**,** y**);**  string day **=** intToDay**(**jd**);**  cout **<<** day **<<** endl**;**  // expected output: 2453089, 3/24/2004, Wed  **}** |

# Geometry

## Formula

* Dist // lines: 
* Normal vector (pháp tuyến): , 
* Direction vector (chỉ phương): 
  1. **Góc giữ 2 vectors:**

|  |  |
| --- | --- |
|  | * Tích chéo giữa 2 vectors      * Tích chéo giữa AB x BC :   > 0 : ***CCW \_ quay ngược chiều kim đồng hồ.***  <0 : ***CW \_ quay thuận chiều kim đồng hồ***.  =0 : **A, B, C thẳng hàng.** |

* 1. **Kiểm tra điểm M có nằm trên đoạn thẳng AB hay không ?**

Điểm M nằm trên đoạn thẳng AB <=> 2 điều kiện sau đều thoả mãn:

1. Điểm M nằm trên đường thẳng AB <=> F (A,B,M) = 0;

2. M nằm giữa Avà B <=> (xm - xa). (xm - xb) <= 0 và (ym - ya). (ym -yb) <= 0

* 1. **Kiểm tra điểm M có thuộc tia AB hay không?**

Điểm M thuộc tia AB <=> 2 điều kiện sau đều thoả mãn:

1. M,A,B thẳng hàng <=> F(A,B,M) = 0.

2. A không nằm giữa B,M <=> (xm - xa). (xb - xa) >= 0 và (ym - ya). (yb - ya) >= 0.

* 1. **Kiểm tra tia AB có cắt đoạn CD không?**

Có thể đưa ra 2 cách giải như sau:

* Cách 1 : Trước tiên tìm giao điểm M (nếu có) của 2 đường thẳng AB và CD, khi đó tia AB cắt đoạn CD <=> 2 điều kiện sau đều thoả mãn:

1. M nằm giữa C, D <=> (xm - xc). (xm - xd) <= 0 và (ym - yc). (ym - yd) <= 0.

2. A không nằm giữa B, M <=> (xm - xa). (xb - xa) >= 0 và (ym - ya). (yb - ya) >= 0.

* Cách 2 : Tia AB cắt đoạn CD <=> 2 điều kiện sau thoả mãn:

1. C, D nằm khác phía so với AB <=> Khacphia (A,B,C,D) = True.

2. A, M nằm khác phía so với CD <=> Khacphia (C,D,A,M) = True. Trong đó M (x,y) là điểm thuộc tia AB và thoả mãn điều kiện:

| xm| > Max (| xc |,|xd |) hoặc |ym | > Max (|yc |,|yd |);

* 1. **Kiểm tra vị trí của một điểm M so với một đa giác** (đa giác có thể lồi hoặc lõm).

|  |  |
| --- | --- |
| Để giải bài toán này, trước hết ta tìm M' (x,y) sao cho x > max { xi : xi là hoành độ một đỉnh i của đa giác ; i=1, 2... } . Còn y có giá trị sao cho đoạn MM' không chứa đỉnh nào của đa giác, y có thể tìm bằng phương pháp duyệt trung điểm của các cạnh hay lựa chọn ngẫu nhiên. Tiếp đó ta đếm số giao điểm của đoạn MM' so với tất cả các cạnh của đa .  + Nếu số giao điểm là lẻ thì M nằm trong đa giác.  + Nếu số giao điểm là chẵn thì M nằm ngoài đa giác.  Chú ý : Nếu M thuộc một cạnh hay một đỉnh nào đó của đa giác thì ta cần xử lý riêng. |  |

* 1. **Tính diện tích tam giác:**

S = (a+b)/2 = (bc *sin*(A) )/2 = sqrt(p(p-a)(p-b)(p-c)) với p=(a+b+c)/2

* 1. **Đường phân giác (P,d):** P = B, 
  2. **Đường tròn nội tiếp tam giác (P,r):**

Incenter: , Radius: 

* 1. **Đường tròn ngoại tiếp tam giác:**

Radius: ****

Circumcentre:

U_x = ((A_x^2 + A_y^2)(B_y - C_y) + (B_x^2 + B_y^2)(C_y - A_y) + (C_x^2 + C_y^2)(A_y - B_y)) / D,

U_y = ((A_x^2 + A_y^2)(C_x - B_x) + (B_x^2 + B_y^2)(A_x - C_x) + (C_x^2 + C_y^2)(B_x - A_x)) / D

with D = 2( A_x(B_y - C_y) + B_x(C_y - A_y) + C_x(A_y - B_y)).\,

* 1. **Tìm 2 điểm gần nhất:**
* B1: Dùng 1 đường thẳng đứng d chia đôi tập hợp Q thành 2 tập hợp Q1 và Q2 sao cho Q1 và Q2 cân bằng.
* B2: Gọi đệ qui tìm khoảng cách ngắn nhất d1 trong Q1 và d2 trong Q2.

d0 = min(d1,d2).

* B3: Tìm tập hợp Q0 bao gồm các điểm cách d một khoảng nhỏ hơn hoặc bằng d0.
* B4: Ứng với mỗi điểm trong Q0 tìm khoảng cách của nó với các điểm thuộc tập đối diện trong khoảng d0
* B5: Trả về khoảng cách ngăn nhất
  1. **Ellipse:**

Point 

1. Ellipse 

Tangent line: 

Normal vector:  (outward) or  (inward)

Guiding vector of tangent line:  (counter-clockwise direction) or  (clockwise direction)

1. Ellipse 

Tangent line: 

* 1. **Tranpezoidal Rule:**

double **f**(int x) {}

double **area**(int a, int b) {

double t = 0;

for (int i = a; i <= b; i++) {

t += f(i);

}

}

* 1. **Circle:**

|  |  |
| --- | --- |
|  | |
| 1. Area   - Sector:  where *n* is the number of degrees in the central angle of the sector,  where CS is the arc length of the sector.  - Segment: | http://www.regentsprep.org/Regents/math/geometry/GP16/segment.gif |
|  |

* 1. **Spere**

|  |  |  |  |
| --- | --- | --- | --- |
| 1. Surface area: 2. Volume: 3. Spherical sector:   - The volume of the spherical sector is: , or  where θ is half the cone angle (the angle between the rim of the cap and the direction to the middle of the cap as seen from the sphere center).  - If the radius of the base of the cap is a, the surface area of the spherical sector is: | | | http://upload.wikimedia.org/wikipedia/commons/4/42/Ausschnitt.PNG |
| 1. Spherical segment   - the volume of the spherical segment is:  - The area of the spherical zone (surface of the segment, excludes the top and bottom bases) is: | | | http://upload.wikimedia.org/wikipedia/commons/thumb/8/83/LaoHaiKugelschicht1.png/320px-LaoHaiKugelschicht1.png |
| 1. Spherical wedge   - Volume:  - Surface area: | http://upload.wikimedia.org/wikipedia/commons/thumb/0/0a/Spherical_Wedge.svg/250px-Spherical_Wedge.svg.png | 1. Spherical cap   - Volume:  - Curved surface area: | http://upload.wikimedia.org/wikipedia/commons/thumb/d/d4/Spherical_Cap.svg/480px-Spherical_Cap.svg.png |

------------------------------------------------------------------------------------------------------

## Hình Giải Tích

* + 1. **Point in 3D:**

|  |  |
| --- | --- |
|  | CROSS PRODUCT    cùng phương |

* + 1. **Plane**

|  |  |
| --- | --- |
| 2 mặt phẳng: | cắt nhau  song song  trùng nhau |

* + 1. **Line in 3D**

1. Phương trình đường thẳng d qua  và có vector chỉ phương :  
    hoặc  với 
2. Tính chất

* 2 đường thẳng  và 
*  cắt nhau 
* 
* 
*  chéo nhau 
  + 1. **Distance in 3D:**

|  |  |
| --- | --- |
| Point to plane: | Point to line: |
| 2 đường thẳng chéo nhau: |  |

* + 1. **Angle in 3D**

1. Between 2 lines:

* 2 đường thẳng  có vector chỉ phương là 
* 

1. Between a line and a plane:

* Đường thẳng d có vector chỉ phương  và mặt phẳng  có vector pháp tuyến 
* 

1. Between 2 planes:



* + 1. **Diện tích, thể tích**

1. Diện tích tam giác ABC: 
2. Diện tích hình bình hành ABCD: 
3. Thể tích hình chóp: 

Thể tích tứ diện ABCD: 

1. Diện tích hình phẳng (tích phân) 1: 
2. Diện tích hình phẳng (tích phân) 2: 
3. Thể tích (tích phân): 
4. Thể tich khối tròn xoay: 

------------------------------------------------------------------------------------------------------

## Tiếp Tuyến

**1. Tiếp tuyến tại một điểm:** y= f(x) (C), x0 thuộc TXĐ của hàm số và tồn tại đạo hàm tại đó 🡪 tiếp tuyến với (C) tại (x0; f(x0)) có phương trình y = y/(x0)(x-x0) + f(x0)

**2. Điều kiện tiếp xúc của hai đồ thị.**

Cho hai hàm số y = f(x) (C1), y = g(x) (C2). (C1) tiếp xúc với (C2) iff có nghiệm.



**3. Tiếp tuyến đi qua một điểm:** Cho hàm số y= f(x) (C) tìm PTTT với (C) đi qua M(xM; yM)

**Cách 1:** Tìm tiếp điểm

Giả sử tiểp tuyến với (C) có tiếp điểm M0(x0; y0) 🡪 tiếp tuyến có phương trình y = f/(x0)(x-x0) + f(x0).

Tiếp tuyến đi qua điểm M(xM; yM) 🡪 yM = f/(x0)(xM-x0) + f(x0) 🡪 giải tìm được hoành độ tiếp điểm 🡪 tìm y0 = f(x0) 🡪 PTTT.

**Cách 2:** Sử dụng điều kiện tiếp xúc

Giả sử đường thẳng qua M(xM; yM) có hệ số góc k 🡪 nó có phương trình

y = k(x-xM) + yM

Ta có đường thẳng y = k(x-xM) + yM là tiếp tuyến của (C) 🡪 giải tìm được hoành độ của tiếp điểm 🡪 PTTT.



**4. Tiếp tuyến cho trước hệ số góc:**

**Cách 1.** Tìm tiếp điểm

Giả sử tiếp tuyến có tiếp điểm M0(x0; y0) 🡪 nó có phương trình y = f/(x0)(x-x0) + f(x0) 🡪 theo giải thiết ta có f/(x0) = k 🡪 Giải tìm được hoành độ tiếp điểm 🡪 tìm y0 = f(x0) 🡪 PTTT.

**Nhận xét:**  Trong dạng này ta có thể gặp các bài tập như sau:

\*) Tiếp tuyến tạo với chiều dương trục hoành góc 🡪 hệ số góc là k = tan 🡪 tìm tiếp điểm M0(x0; y0) bằng cách giải f/(x0) = k 🡪 PTTT.



\*) Tiếp tuyến tạo với y = ax +b một góc 🡪 hệ số góc là k thoả hoặc dùng tích vô hướng của 2 véctơ pháp tuyến để tìm k 🡪 tìm tiếp điểm M0(x0; y0) bằng cách f/(x0) = k 🡪 PTTT.



## Standford Geometry:

double INF = 1e100;

double EPS = 1e-12;

struct PT {

double x, y;

**PT**() {}

**PT**(double x, double y) : x(x), y(y) {}

**PT**(const PT &p) : x(p.x), y(p.y) {}

PT **operator** **+** (const PT &p) const { return PT(x+p.x, y+p.y); }

PT **operator** **-** (const PT &p) const { return PT(x-p.x, y-p.y); }

PT **operator** **\*** (double c) const { return PT(x\*c, y\*c ); }

PT **operator** **/** (double c) const { return PT(x/c, y/c ); }

};

double **dot**(PT p, PT q) { return p.x\*q.x+p.y\*q.y; }

double **dist2**(PT p, PT q) { return dot(p-q,p-q); }

double **cross**(PT p, PT q) { return p.x\*q.y-p.y\*q.x; }

ostream &**operator<<**(ostream &os, const PT &p) {

os << **"**(**"** << p.x << **"**,**"** << p.y << **"**)**"**;

}

// rotate a point CCW or CW around the origin

PT **RotateCCW90**(PT p) { return PT(-p.y,p.x); }

PT **RotateCW90**(PT p) { return PT(p.y,-p.x); }

PT **RotateCCW**(PT p, double t) {

return PT(p.x\*cos(t)-p.y\*sin(t), p.x\*sin(t)+p.y\*cos(t));

}

// project point c onto line through a and b assuming a != b

PT **ProjectPointLine**(PT a, PT b, PT c) {

return a + (b-a)\*dot(c-a, b-a)/dot(b-a, b-a);

}

// project point c onto line segment through a and b

PT **ProjectPointSegment**(PT a, PT b, PT c) {

double r = dot(b-a,b-a);

if (fabs(r) < EPS) return a;

r = dot(c-a, b-a)/r;

if (r < 0) return a;

if (r > 1) return b;

return a + (b-a)\*r;

}

// compute distance from c to segment between a and b

double **DistancePointSegment**(PT a, PT b, PT c) {

return sqrt(dist2(c, ProjectPointSegment(a, b, c)));

}

// compute distance between point (x,y,z) and plane ax+by+cz=d

double **DistancePointPlane**(double x, double y, double z,

double a, double b, double c, double d) {

return fabs(a\*x+b\*y+c\*z-d)/sqrt(a\*a+b\*b+c\*c);

}

// determine if lines from a to b and c to d are parallel or collinear

bool **LinesParallel**(PT a, PT b, PT c, PT d) {

return fabs(cross(b-a, c-d)) < EPS;

}

bool **LinesCollinear**(PT a, PT b, PT c, PT d) {

return LinesParallel(a, b, c, d)

&& fabs(cross(a-b, a-c)) < EPS

&& fabs(cross(c-d, c-a)) < EPS;

}

// determine if line segment from a to b intersects with line segment from c to d

bool **SegmentsIntersect**(PT a, PT b, PT c, PT d) {

if (LinesCollinear(a, b, c, d)) {

if (dist2(a, c) < EPS || dist2(a, d) < EPS ||

dist2(b, c) < EPS || dist2(b, d) < EPS) return true;

if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-b) > 0)

return false;

return true;

}

if (cross(d-a, b-a) \* cross(c-a, b-a) > 0) return false;

if (cross(a-c, d-c) \* cross(b-c, d-c) > 0) return false;

return true;

}

// compute intersection of line passing through a and b with line passing through c and d, assuming that unique intersection exists; for segment intersection, check if segments intersect first

PT **ComputeLineIntersection**(PT a, PT b, PT c, PT d) {

b=b-a; d=c-d; c=c-a;

assert(dot(b, b) > EPS && dot(d, d) > EPS);

return a + b\*cross(c, d)/cross(b, d);

}

// compute center of circle given three points

PT **ComputeCircleCenter**(PT a, PT b, PT c) {

b=(a+b)/2;

c=(a+c)/2;

return ComputeLineIntersection(b, b+RotateCW90(a-b), c, c+RotateCW90(a-c));

}

// determine if point is in a possibly non-convex polygon (by William Randolph Franklin); returns 1 for strictly interior points, 0 for strictly exterior points, and 0 or 1 for the remaining points. Note that it is possible to convert this into an \*exact\* test using integer arithmetic by taking care of the division appropriately (making sure to deal with signs properly) and then by writing exact tests for checking point on polygon boundary

bool **PointInPolygon**(const vector<PT> &p, PT q) {

bool c = 0;

for (int i = 0; i < p.size(); i++){

int j = (i+1)%p.size();

if ((p[i].y <= q.y && q.y < p[j].y ||

p[j].y <= q.y && q.y < p[i].y) &&

q.x < p[i].x + (p[j].x - p[i].x) \* (q.y - p[i].y) / (p[j].y - p[i].y))

c = !c;

}

return c;

}

// determine if point is on the boundary of a polygon

bool **PointOnPolygon**(const vector<PT> &p, PT q) {

for (int i = 0; i < p.size(); i++)

if (dist2(ProjectPointSegment(p[i], p[(i+1)%p.size()], q), q) < EPS)

return true;

return false;

}

// compute intersection of line through points a and b with circle centered at c with radius r > 0

vector<PT> **CircleLineIntersection**(PT a, PT b, PT c, double r) {

vector<PT> ret;

b = b-a;

a = a-c;

double A = dot(b, b);

double B = dot(a, b);

double C = dot(a, a) - r\*r;

double D = B\*B - A\*C;

if (D < -EPS) return ret;

ret.push\_back(c+a+b\*(-B+sqrt(D+EPS))/A);

if (D > EPS)

ret.push\_back(c+a+b\*(-B-sqrt(D))/A);

return ret;

}

// compute intersection of circle centered at a with radius r with circle centered at b with radius R

vector<PT> **CircleCircleIntersection**(PT a, PT b, double r, double R) {

vector<PT> ret;

double d = sqrt(dist2(a, b));

if (d > r+R || d+min(r, R) < max(r, R)) return ret;

double x = (d\*d-R\*R+r\*r)/(2\*d);

double y = sqrt(r\*r-x\*x);

PT v = (b-a)/d;

ret.push\_back(a+v\*x + RotateCCW90(v)\*y);

if (y > 0)

ret.push\_back(a+v\*x - RotateCCW90(v)\*y);

return ret;

}

// This code computes the area or centroid of a (possibly nonconvex) polygon, assuming that the coordinates are listed in a clockwise or counterclockwise fashion. Note that the centroid is often known as the "center of gravity" or "center of mass".

double **ComputeSignedArea**(const vector<PT> &p) {

double area = 0;

for(int i = 0; i < p.size(); i++) {

int j = (i+1) % p.size();

area += p[i].x\*p[j].y - p[j].x\*p[i].y;

}

return area / 2.0;

}

double **ComputeArea**(const vector<PT> &p) {

return fabs(ComputeSignedArea(p));

}

PT **ComputeCentroid**(const vector<PT> &p) {

PT c(0,0);

double scale = 6.0 \* ComputeSignedArea(p);

for (int i = 0; i < p.size(); i++){

int j = (i+1) % p.size();

c = c + (p[i]+p[j])\*(p[i].x\*p[j].y - p[j].x\*p[i].y);

}

return c / scale;

}

// tests whether or not a given polygon (in CW or CCW order) is simple

bool **IsSimple**(const vector<PT> &p) {

for (int i = 0; i < p.size(); i++) {

for (int k = i+1; k < p.size(); k++) {

int j = (i+1) % p.size();

int l = (k+1) % p.size();

if (i == l || j == k) continue;

if (SegmentsIntersect(p[i], p[j], p[k], p[l]))

return false;

}

}

return true;

}

double **cross\_product**(PT a, PT b, PT c) { // also 2 \* signed triangle area

return (a.x \* b.y - a.y \* b.x + a.y \* c.x - a.x \* c.y + b.x \* c.y – c.x \* b.y);

// same as return ((b.x - a.x) \* (c.y - a.y) - (b.y - a.y) \* (c.x - a.x));

//Counter-clockwise > DBL\_EPSILON

//Clockwise **< DBL\_EPSILON**

//Collinear fabs() **<= DBL\_EPSILON**

}

bool **lexico\_compare**(PT i, PT j) {

return (i.x < j.x || (i.x == j.x && i.y < j.y));

}

vector<point> **Andrew\_convex\_hull**(vector< PT > P) {

vector<point> hull(2 \* P.size());

double sum;

int l\_size, u\_size, i;

sort(P.begin(), P.end(), lexico\_compare);

// Build lower hull

l\_size = 0;

for (i = 0; i < P.size(); i++) {

while (l\_size >= 2 && cross\_product(hull[l\_size - 2], hull[l\_size - 1], P[i]) <= DBL\_EPSILON) l\_size--;

hull[l\_size] = P[i];

l\_size++;

}

l\_size--; // Remove last point;

// Build upper hull

u\_size = l\_size;

for (i = P.size() - 1; i >= 0; i--) {

while (u\_size >= l\_size + 2 && cross\_product(hull[u\_size - 2], hull[u\_size - 1], P[i]) <= DBL\_EPSILON) u\_size--;

hull[u\_size] = P[i];

u\_size++;

}

u\_size--; // Remove last point;

hull.resize(u\_size);

return hull;

}

int **main**() {

// expected: (-5,2)

cerr << RotateCCW90(PT(2,5)) << endl;

// expected: (5,-2)

cerr << RotateCW90(PT(2,5)) << endl;

// expected: (-5,2)

cerr << RotateCCW(PT(2,5),M\_PI/2) << endl;

// expected: (5,2)

cerr << ProjectPointLine(PT(-5,-2), PT(10,4), PT(3,7)) << endl;

// expected: (5,2) (7.5,3) (2.5,1)

cerr << ProjectPointSegment(PT(-5,-2), PT(10,4), PT(3,7)) << **"** **"**

<< ProjectPointSegment(PT(7.5,3), PT(10,4), PT(3,7)) << **"** **"**

<< ProjectPointSegment(PT(-5,-2), PT(2.5,1), PT(3,7)) << endl;

// expected: 6.78903

cerr << DistancePointPlane(4,-4,3,2,-2,5,-8) << endl;

// expected: 1 0 1

cerr << LinesParallel(PT(1,1), PT(3,5), PT(2,1), PT(4,5)) << **"** **"**

<< LinesParallel(PT(1,1), PT(3,5), PT(2,0), PT(4,5)) << **"** **"**

<< LinesParallel(PT(1,1), PT(3,5), PT(5,9), PT(7,13)) << endl;

// expected: 0 0 1

cerr << LinesCollinear(PT(1,1), PT(3,5), PT(2,1), PT(4,5)) << **"** **"**

<< LinesCollinear(PT(1,1), PT(3,5), PT(2,0), PT(4,5)) << **"** **"**

<< LinesCollinear(PT(1,1), PT(3,5), PT(5,9), PT(7,13)) << endl;

// expected: 1 1 1 0

cerr << SegmentsIntersect(PT(0,0), PT(2,4), PT(3,1), PT(-1,3)) << **"** **"**

<< SegmentsIntersect(PT(0,0), PT(2,4), PT(4,3), PT(0,5)) << **"** **"**

<< SegmentsIntersect(PT(0,0), PT(2,4), PT(2,-1), PT(-2,1)) << **"** **"**

<< SegmentsIntersect(PT(0,0), PT(2,4), PT(5,5), PT(1,7)) << endl;

// expected: (1,2)

cerr << ComputeLineIntersection(PT(0,0), PT(2,4), PT(3,1), PT(-1,3)) << endl;

// expected: (1,1)

cerr << ComputeCircleCenter(PT(-3,4), PT(6,1), PT(4,5)) << endl;

vector<PT> v;

v.push\_back(PT(0,0));

v.push\_back(PT(5,0));

v.push\_back(PT(5,5));

v.push\_back(PT(0,5));

// expected: 1 1 1 0 0

cerr << PointInPolygon(v, PT(2,2)) << **"** **"**

<< PointInPolygon(v, PT(2,0)) << **"** **"**

<< PointInPolygon(v, PT(0,2)) << **"** **"**

<< PointInPolygon(v, PT(5,2)) << **"** **"**

<< PointInPolygon(v, PT(2,5)) << endl;

// expected: 0 1 1 1 1

cerr << PointOnPolygon(v, PT(2,2)) << **"** **"**

<< PointOnPolygon(v, PT(2,0)) << **"** **"**

<< PointOnPolygon(v, PT(0,2)) << **"** **"**

<< PointOnPolygon(v, PT(5,2)) << **"** **"**

<< PointOnPolygon(v, PT(2,5)) << endl;

// expected: (1,6)

// (5,4) (4,5)

// blank line

// (4,5) (5,4)

// blank line

// (4,5) (5,4)

vector<PT> u = CircleLineIntersection(PT(0,6), PT(2,6), PT(1,1), 5);

for (int i = 0; i < u.size(); i++) cerr << u[i] << **"** **"**; cerr << endl;

u = CircleLineIntersection(PT(0,9), PT(9,0), PT(1,1), 5);

for (int i = 0; i < u.size(); i++) cerr << u[i] << **"** **"**; cerr << endl;

u = CircleCircleIntersection(PT(1,1), PT(10,10), 5, 5);

for (int i = 0; i < u.size(); i++) cerr << u[i] << **"** **"**; cerr << endl;

u = CircleCircleIntersection(PT(1,1), PT(8,8), 5, 5);

for (int i = 0; i < u.size(); i++) cerr << u[i] << **"** **"**; cerr << endl;

u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 10, sqrt(2.0)/2.0);

for (int i = 0; i < u.size(); i++) cerr << u[i] << **"** **"**; cerr << endl;

u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 5, sqrt(2.0)/2.0);

for (int i = 0; i < u.size(); i++) cerr << u[i] << **"** **"**; cerr << endl;

// area should be 5.0 centroid should be (1.1666666, 1.166666)

PT pa[] = { PT(0,0), PT(5,0), PT(1,1), PT(0,5) };

vector<PT> p(pa, pa+4);

PT c = ComputeCentroid(p);

cerr << **"**Area: **"** << ComputeArea(p) << endl;

cerr << **"**Centroid: **"** << c << endl;

return 0;

}

## Geo3D

**public** **class** Geom3D {

*// distance from point (x, y, z) to plane aX + bY + cZ + d = 0*

**public** **static** **double** ptPlaneDist(**double** x, **double** y, **double** z, **double** a, **double** b, **double** c,

**double** d) {

**return** Math.abs(a\*x + b\*y + c\*z + d) / Math.sqrt(a\*a + b\*b + c\*c);

}

*// distance between parallel planes aX + bY + cZ + d1 = 0 and aX + bY + cZ + d2 = 0*

**public** **static** **double** planePlaneDist(**double** a, **double** b, **double** c, **double** d1, **double** d2) {

**return** Math.abs(d1 - d2) / Math.sqrt(a\*a + b\*b + c\*c);

}

*// distance from point (px, py, pz) to line (x1, y1, z1)-(x2, y2, z2) (or ray, or segment; in*

*// the case of the ray, the endpoint is the first point)*

**public** **static** **final** **int** LINE = 0;

**public** **static** **final** **int** SEGMENT = 1;

**public** **static** **final** **int** RAY = 2;

**public** **static** **double** ptLineDistSq(**double** x1, **double** y1, **double** z1, **double** x2, **double** y2,

**double** z2, **double** px, **double** py, **double** pz,**int** type) {

**double** pd2 = (x1-x2)\*(x1-x2) + (y1-y2)\*(y1-y2) + (z1-z2)\*(z1-z2);

**double** x, y, z;

**if** (pd2 == 0) {

x = x1;

y = y1;

z = z1;

} **else** {

**double** u = ((px-x1)\*(x2-x1) + (py-y1)\*(y2-y1) + (pz-z1)\*(z2-z1)) / pd2;

x = x1 + u \* (x2 - x1);

y = y1 + u \* (y2 - y1);

z = z1 + u \* (z2 - z1);

**if** (type != LINE && u < 0) {

x = x1;

y = y1;

z = z1;

}

**if** (type == SEGMENT && u > 1.0) {

x = x2;

y = y2;

z = z2;

}

}

**return** (x-px)\*(x-px) + (y-py)\*(y-py) + (z-pz)\*(z-pz);

}

**public** **static** **double** ptLineDist(**double** x1, **double** y1, **double** z1, **double** x2, **double** y2,

**double** z2, **double** px, **double** py, **double** pz, **int** type) {

**return** Math.sqrt(ptLineDistSq(x1, y1, z1, x2, y2, z2, px, py, pz, type));

}

}

# Graph

## FastDijkstra.cc

*// Implementation of Dijkstra's algorithm using adjacency lists and priority queue for*

*// efficiency. Running time: O(|E| log |V|)*

#definde SIZE 100000 + 1

vector < pair < int , int > > v [SIZE]; // each vertex has all the connected vertices with the edges weights

int dist [SIZE];

bool vis [SIZE];

void dijkstra(){

// set the vertices distances as infinity

memset(vis, false , sizeof vis); // set all vertex as unvisited

dist[1] = 0;

multiset < pair < int , int > > s; // multiset do the job as a min-priority queue

s.insert({0 , 1}); // insert the source node with distance = 0

while(!s.empty()){

pair <int , int> p = \*s.begin(); // pop the vertex with the minimum distance

s.erase(s.begin());

int x = p.s; int wei = p.f;

if( vis[x] ) continue; // check if the popped vertex is visited before

vis[x] = true;

for(int i = 0; i < v[x].size(); i++){

int e = v[x][i].f; int w = v[x][i].s;

if(dist[x] + w < dist[e] ){ // check if the next vertex distance could be minimized

dist[e] = dist[x] + w;

s.insert({dist[e], e} ); // insert the next vertex with the updated distance

}

}

}

}

}

## Bellman\_Ford

// The main function that finds shortest distances from src to all other vertices using Bellman-Ford

// algorithm. The function also detects negative weight cycle and print out the list of distance from

// source to the rest vertex

vector <int> v [2000 + 10];

int dis [1000 + 10];

for(int i = 0; i < m + 2; i++){

v[i].clear();

dis[i] = 2e9;

}

for(int i = 0; i < m; i++){

scanf("%d%d%d", &from , next , &weight);

v[i].push\_back(from);

v[i].push\_back(next);

v[i].push\_back(weight);

}

dis[0] = 0;

for(int i = 0; i < n - 1; i++){

int j = 0;

while(v[j].size() != 0){

if(dis[ v[j][0] ] + v[j][2] < dis[ v[j][1] ] ){

dis[ v[j][1] ] = dis[ v[j][0] ] + v[j][2];

}

j++;

}

}

## Floyd\_Warshall

const int maxN = 1001;

int next[maxN][maxN];

int a[maxN][maxN];

void **floyd\_warshall\_trace**(int i, int j) {

int mid = next[i][j];

if (next[i][j] != 0) {

floyd\_warshall\_trace(i, next[i][j]);

cout << mid << **"** **"**;

floyd\_warshall\_trace(next[i][j], j);

}

}

int **main**() {

// Initialize

int vertices = 3;

// initialize diagonal

for (int i = 1; i <= vertices; i++)

for (int j = 1; j <= vertices; j++)

a[i][j] = INT\_MAX;

// initialize distances

a[1][2] = 20;

a[2][3] = 10; //……

// Floyd-Warshall

// Add nodes between (first 1 then 2, 3 till n) and look if distance is shorter

for (int k = 1; k <= vertices; k++)

for (int i = 1; i <= vertices; i++)

for (int j = 1; j <= vertices; j++)

if (a[i][j] > a[i][k] + a[k][j] && a[i][k] != INT\_MAX && a[k][j] != INT\_MAX) {

a[i][j] = a[i][k] + a[k][j];

next[i][j] = k;

}

// Print out shortest path from u to v

int u = 1, v = 3;

cout << **"**Shortest path from **"** << u << **"** to **"** << v << **"** is **"** << a[u][v] << endl;

cout << u << **"** **"**;

floyd\_warshall\_trace(u, v);

cout << v << endl;

return 0;

}

## Euler\_Path

#include <iostream>

#include <list>

int graph[101][101];

int n, m, x, y, steps;

list<int> path;

void **walk**(int pos) {

for (int i = 1; i <= n; i++) {

if (graph[pos][i] > 0) {

graph[pos][i]--;

walk(i);

break;

}

}

path.push\_back(pos + 1);

}

int **main**() {

cin >> n >> m;

// Ham input su dung cho do thi co huong

for (int i = 0; i < m; i++) {

cin >> x >> y;

graph[x][y]++;

}

walk(1);

while (!path.empty()) {

cout << path.back() << ' ';

path.pop\_back();

}

}

## Hamiltonian\_Path/Circuit

const int maxN = 1001;

bool arr[maxN][maxN], avai[maxN], found;

int x[maxN]; // array luu giu thu tu cac dinh trong duong di tim duoc, tu x[1] -> x[n]

int n, m;

void **enter**() {

// Khoi tao du lieu ban dau

cin >> n >> m;

for (int i = 1; i <= n; i++) {

// Khoi tao mang avai[]

avai[i] = true;

// Khoi tao ma tran arr[][]

for (int j = 1; j <= n; j++) {

arr[u][v] = false;

}

}

// Danh dau dinh 1 la dinh bat dau

avai[1] = false;

// Nhap m canh cho graph

for (int i = 0; i < m; i++) {

int u, v;

cin >> u >> v;

arr[u][v] = arr[v][u] = true;

}

found = false;

x[1] = 1; // Gan vi tri bat dau cua duong di la dinh 1

}

// Thuat toan quay lui, lan goi dau tien i = 2

void **attemp**(int i) {

for (int v = 2; v <= n; v++) {

// Neu da tim duoc duong di thi exit

if (found) return;

// Xet cac dinh v chua noi qua va ke voi x[i-1]

if (avai[v] && arr[x[i - 1]], [v]) {

x[i] = v; // Thu noi sang v

if (i == n) { // Neu da noi qua du n dinh

// Neu dinh thu n quay ve dinh 0 thi duoc circuit

if (arr[v][1]) found = true;

return;

} else { // Chua qua du n dinh

avai[v] = false; // Danh dau dinh v la dinh da di qua

attemp(i + 1); // di tiep

if (found) return; // Neu tim ra duoc circuit thi thoat

avai[v] = true; // Xoa danh dau dinh V

}

}

}

}

## Prim(MSTree)

#define typec int // type of cost

struct Edge { int e1; int e2;} edge[10000];

const typec inf = 0x3f3f3f3f; // max of cost

const int V = 100;

int vis[V], totalE = 0, truoc[V];

typec lowc[V];

typec **prim**(typec cost[V][V], int n) // vertex: 0 ~ n-1 {

int i, j, p;

typec minc, res = 0;

memset(vis, 0, sizeof (vis));

vis[0] = 1; // Danh dau dinh dau tien da duoc xet

// Khoi tao khoang cach ngan nhat tu cay khung den cac dinh chua xet

for (i = 1; i < n; i++) {

lowc[i] = cost[0][i];

truoc[i] = 0; // Khoi tao gia tri truoc[] = index cua dinh dau tien

}

for (i = 1; i < n; i++) {

minc = inf;

p = -1;

for (j = 0; j < n; j++)

if (0 == vis[j] && minc > lowc[j]) {

minc = lowc[j];

p = j;

}

if (inf == minc) return -1; // do thi khong lien thong, thoat khoi chuong trinh

res += minc;

vis[p] = 1; // Cap nhat chieu dai cay khung

// Cap nhat canh cua cay khung

edge[totalE].e1 = truoc[p];

edge[totalE].e2 = p;

totalE++;

//Cap nhat lai khoang cach ngan nhat tu cay khung den cac dinh chua xet

for (j = 0; j < n; j++)

if (0 == vis[j] && lowc[j] > cost[p][j]) {

lowc[j] = cost[p][j];

truoc[j] = p;

}

}

return res;

}

int **main**() {

typec g[V][V];

g[0][1] = g[1][0] = 2;

g[0][2] = g[2][0] = 6;

g[1][2] = g[2][1] = 3;

typec res = prim(g, 3);

cout << res << endl;

for (int i = 0; i < totalE; i++) {

cout << edge[i].e1 << **"** **"** << edge[i].e2 << **"** ----> **"**;

}

return 0;

}

## Tarjan (Liet ke tat ca thanh phan lien thong manh).

const int maxN = 1000;

vector<int> dinh(maxN, 0), label(maxN, 0), number(maxN, 0), low(maxN, INT\_MAX);

vector<int> Q;

int count, n, total, a[maxN][maxN];

void **tarjan**(int u) { // vertex: 0 ~ n-1

int v;

count++;

number[u] = count;

Q.push\_back(u); // Day u vao stack

for (v = 1; v <= n; v++) {

if (label[v] == 0) { // Neu dinh v con nam trong do thi

if (a[u][v] == 1) {

if (number[v] > 0) { // Neu v da tham

low[u] = min(low[u], number[v]); // cap nhat dinh tien boi cua u

} else { // Chua tham

tarjan(v);

low[u] = min(low[u], low[v]); // cap nhat dinh tien boi cua u

}

}

}

}

// Khi da duyet xong u

if (low[u] >= number[u]) { // Neu u la chot cua 1 thanh phan lien thong

total++;

cout << **"**Thanh phan lien thong thu **"** << total << endl;

v = Q.back();

if (u != v) cout << u << **"** <-- **"**;

while (1) {

v = Q.back();

Q.pop\_back();

cout << v;

label[v] = 1; // loai dinh v ra khoi do thi

if (u != v) cout << **"** <-- **"**;

else {

cout << endl;

break;

}

}

}

}

int **main**() {

count = total = 0;

// Khoi tao graph

n = 7;

a[1][2] = a[2][3] = a[3][4] = a[4][2] = a[4][5] = 1;

a[5][6] = a[6][7] = a[7][5] = 1;

for (int i = 1; i <= n; i++) {

if (number[i] == 0) tarjan(i);

}

return 0;

}

## Minimum Cost Bipartite Matching: Hungarian algorithm

// Bai toan chia N cong nhan lam M cong viec voi cost nho nhat

#include <queue>

#include <climits>

#define maxN 155

#define maxV INT\_MAX

using namespace std;

int c[maxN][maxN], Fx[maxN], Fy[maxN];

int matchX[maxN], matchY[maxN], trace[maxN];

int n, m, k, start, finish;

void **loadData**() {

int i, j, z;

cin >> n >> m >> z;

k = max(n, m);

for (i = 1; i <= k; i++)

for (j = 1; j <= k; j++) c[i][j] = maxV;

while (z--) {

cin >> i >> j;

cin >> c[i][j];

}

}

int **getC**(int i, int j) {

return (c[i][j] - Fx[i] - Fy[j]);

}

void **find\_augmenting\_path**() {

queue<int> q;

int i, j;

fill(trace + 1, trace + k + 1, 0);

q.push(start);

while (!q.empty()) {

i = q.front();

q.pop();

for (j = 1; j <= k; j++)

if (trace[j] == 0 && getC(i, j) == 0) {

trace[j] = i;

if (matchY[j] == 0) {

finish = j;

return;

}

q.push(matchY[j]);

}

}

}

void **subX\_addY**() {

int i, j, t, d;

bool visitedX[k + 1];

bool visitedY[k + 1];

fill(visitedX + 1, visitedX + k + 1, false);

fill(visitedY + 1, visitedY + k + 1, false);

visitedX[start] = true;

for (j = 1; j <= k; j++)

if (trace[j] != 0) {

visitedX[matchY[j]] = true;

visitedY[j] = true;

}

d = maxV;

for (i = 1; i <= k; i++)

if (visitedX[i])

for (j = 1; j <= k; j++)

if (!visitedY[j] && getC(i, j) < d)

d = getC(i, j);

for (t = 1; t <= k; t++) {

if (visitedX[t]) Fx[t] += d;

if (visitedY[t]) Fy[t] -= d;

}

}

void **enlarge**() {

int i, next;

while (finish != 0) {

i = trace[finish];

next = matchX[i];

matchX[i] = finish;

matchY[finish] = i;

finish = next;

}

}

void **solve**() {

int i, j, l, cost;

fill(matchX + 1, matchX + k + 1, 0);

fill(matchY + 1, matchY + k + 1, 0);

fill(Fx + 1, Fx + k + 1, 0);

fill(Fy + 1, Fy + k + 1, 0);

for (i = 1; i <= k; i++) {

start = i;

finish = 0;

while (finish == 0) {

find\_augmenting\_path();

if (finish == 0) subX\_addY();

}

enlarge();

}

// Print result

cost = 0;

for (i = 1; i <= k; i++) {

j = matchX[i];

if (c[i][j] < maxV) {

cout << **"**x[**"** << i << **"**] <-> y[**"** << j << **"**]: **"** << c[i][j] << **"**; **"**;

cost += c[i][j];

}

}

cout << **"\n**Total Cost**"** << cost << endl;

}

int **main**(int argc, char\*\* argv) {

loadData();

solve();

return 0;

}

## Max Flow: Ford-Fulkerson algorithm

#define maxN 1001

#define maxV INT\_MAX

int c[maxN][maxN];

int f[maxN][maxN];

int trace[maxN];

int n, s, t;

void **loadData**() {

int i, j, z;

cin >> n >> z >> s >> t;

while (z--) {

cin >> i >> j;

cin >> c[i][j];

}

}

bool **find\_path**() { // Tim duong tang luong

int u, v;

queue<int> q;

fill(trace + 1, trace + n + 1, -1);

trace[s] = n + 1;

q.push(s);

while (!q.empty()) {

u = q.front();

q.pop();

for (v = 1; v <= n; v++)

if (trace[v] == -1 && c[u][v] > f[u][v]) {

trace[v] = u;

if (v == t) return true;

q.push(v);

}

}

return false;

}

void **increase\_flow**() { // Tang luong doc duong tang luong

int u, v, d;

d = maxV;

v = t;

while (v != s) {

u = trace[v];

d = min(d, c[u][v] - f[u][v]);

v = u;

}

// f = f + fp

v = t;

while (v != s) {

u = trace[v];

f[u][v] += d;

f[v][u] -= d;

v = u;

}

}

void **solve**() {

int i, j, m;

for (i = 1; i <= n; i++)

for (j = 1; j <= n; j++)

f[i][j] = 0;

while (find\_path())

increase\_flow();

// Print result

m = 0;

for (i = 1; i <= n; i++)

for (j = 1; j <= n; j++)

if (f[i][j] > 0) {

cout << **"**f[**"** << i << **"**][**"** << j << **"**] = **"** << f[i][j] << endl;

if (i == s)

m += f[s][j];

}

cout << **"**Max Flow: **"** << m;

}

int **main**(int argc, char\*\* argv) {

loadData();

solve();

return 0;

}

## Maximum Matching

// Tim cach bat cap nhieu nhat giua 2 tap hop, z la tong so moi lien he giua 2 tap hop

#define maxN 105

using namespace std;

bool a[maxN][maxN];

int matchX[maxN], matchY[maxN], trace[maxN];

int m, n;

void **loadData**() {

int i, j, z;

cin >> m >> n >> z;

for (i = 1; i <= m; i++)

for (j = 1; j <= n; j++)

a[i][j] = false;

while (z--) {

cin >> i >> j;

a[i][j] = true;

}

}

int **find\_augmenting\_path**() {

// Tim duong mo, neu thay tra ve mot Y-dinh chua ghep la dinh ket thuc duong mo, neu khong thay tra ve 0

queue<int> q;

int i, j;

fill(trace + 1, trace + n + 1, 0);

for (i = 1; i <= m; i++)

if (matchX[i] == 0) q.push(i);

while (!q.empty()) {

i = q.front();

q.pop();

for (j = 1; j <= n; j++)

if (trace[j] == 0 && a[i][j] && matchX[i] != j) {

trace[j] = i;

if (matchY[j] == 0)

return j;

q.push(matchY[j]);

}

}

return 0;

}

void **enlarge**(int f) { // Mo rong bo ghep bang duong mo ket thuc o f thuoc Y

int x, next;

while (f != 0) {

x = trace[f];

next = matchX[x];

matchX[x] = f;

matchY[f] = x;

f = next;

}

}

void **solve**() {

int finish, count;

fill(matchX + 1, matchX + m + 1, 0);

fill(matchY + 1, matchY + n + 1, 0);

while (finish = find\_augmenting\_path())

enlarge(finish);

count = 0;

for (int i = 1; i <= m; i++)

if (matchX[i] != 0) {

count++;

cout << **"**x[**"** << i << **"**] - y[**"** << matchX[i] << **"**]**"** << endl;

}

}

int **main**(int argc, char\*\* argv) {

loadData();

solve();

return 0;

}

## Dinic.cc

*// Adjacency list implementation of Dinic's blocking flow algorithm. This is very fast in*

*// practice, and only loses to push-relabel flow.*

*// Running time: O(|V|^2 |E|)*

*// INPUT:*

*// - graph, constructed using AddEdge()*

*// - source*

*// - sink*

*// OUTPUT:*

*// - maximum flow value*

*// - To obtain the actual flow values, look at all edges with*

*// capacity > 0 (zero capacity edges are residual edges).*

using namespace std;

**const** **int** INF = 2000000000;

**struct** Edge {

**int** from, to, cap, flow, index;

Edge(**int** from, **int** to, **int** cap, **int** flow, **int** index) :

from(from), to(to), cap(cap), flow(flow), index(index) {}

};

**struct** Dinic {

**int** N;

vector<vector<Edge> > G;

vector<Edge \*> dad;

vector<**int**> Q;

Dinic(**int** N) : N(N), G(N), dad(N), Q(N) {}

**void** AddEdge(**int** from, **int** to, **int** cap) {

G[from].push\_back(Edge(from, to, cap, 0, G[to].size()));

**if** (from == to) G[from].back().index++;

G[to].push\_back(Edge(to, from, 0, 0, G[from].size() - 1));

}

**long** **long** BlockingFlow(**int** s, **int** t) {

fill(dad.begin(), dad.end(), (Edge \*) NULL);

dad[s] = &G[0][0] - 1;

**int** head = 0, tail = 0;

Q[tail++] = s;

**while** (head < tail) {

**int** x = Q[head++];

**for** (**int** i = 0; i < G[x].size(); i++) {

Edge &e = G[x][i];

**if** (!dad[e.to] && e.cap - e.flow > 0) {

dad[e.to] = &G[x][i];

Q[tail++] = e.to;

}

}

}

**if** (!dad[t]) **return** 0;

**long** **long** totflow = 0;

**for** (**int** i = 0; i < G[t].size(); i++) {

Edge \*start = &G[G[t][i].to][G[t][i].index];

**int** amt = INF;

**for** (Edge \*e = start; amt && e != dad[s]; e = dad[e->from]) {

**if** (!e) { amt = 0; **break**; }

amt = min(amt, e->cap - e->flow);

}

**if** (amt == 0) **continue**;

**for** (Edge \*e = start; amt && e != dad[s]; e = dad[e->from]) {

e->flow += amt;

G[e->to][e->index].flow -= amt;

}

totflow += amt;

}

**return** totflow;

}

**long** **long** GetMaxFlow(**int** s, **int** t) {

**long** **long** totflow = 0;

**while** (**long** **long** flow = BlockingFlow(s, t)) totflow += flow;

**return** totflow;

}

};

int main(){

int n, m, s, t;

int a, b, c;

cin >> n >> m >> s >> t;

Dinic dinic(m);

while (m--){

cin >> a >> b >> c;

dinic.AddEdge(a, b, c);

}

cout << dinic.GetMaxFlow(s, t) << endl;

return 0;

}

## MinCostMaxFlow.cc

*// Implementation of min cost max flow algorithm using adjacency matrix (Edmonds and Karp 1972).*

*// This implementation keeps track of forward and reverse edges separately (so you can set*

*// cap[i][j] != cap[j][i]). For a regular max flow, set all edge costs to 0. Running time,*

*// O(|V|^2) cost per augmentation*

*// max flow: O(|V|^3) augmentations*

*// min cost max flow: O(|V|^4 \* MAX\_EDGE\_COST) augmentations*

*// INPUT:*

*// - graph, constructed using AddEdge()*

*// - source*

*// - sink*

*// OUTPUT:*

*// - (maximum flow value, minimum cost value)*

*// - To obtain the actual flow, look at positive values only.*

**typedef** vector<**int**> VI;

**typedef** vector<VI> VVI;

**typedef** **long** **long** L;

**typedef** vector<L> VL;

**typedef** vector<VL> VVL;

**typedef** pair<**int**, **int**> PII;

**typedef** vector<PII> VPII;

**const** L INF = numeric\_limits<L>::max() / 4;

**struct** MinCostMaxFlow {

**int** N;

VVL cap, flow, cost;

VI found;

VL dist, pi, width;

VPII dad;

MinCostMaxFlow(**int** N) :

N(N), cap(N, VL(N)), flow(N, VL(N)), cost(N, VL(N)),

found(N), dist(N), pi(N), width(N), dad(N) {}

**void** AddEdge(**int** from, **int** to, L cap, L cost) {

**this**->cap[from][to] = cap;

**this**->cost[from][to] = cost;

}

**void** Relax(**int** s, **int** k, L cap, L cost, **int** dir) {

L val = dist[s] + pi[s] - pi[k] + cost;

**if** (cap && val < dist[k]) {

dist[k] = val;

dad[k] = make\_pair(s, dir);

width[k] = min(cap, width[s]);

}

}

L Dijkstra(**int** s, **int** t) {

fill(found.begin(), found.end(), false);

fill(dist.begin(), dist.end(), INF);

fill(width.begin(), width.end(), 0);

dist[s] = 0;

width[s] = INF;

**while** (s != -1) {

**int** best = -1;

found[s] = true;

**for** (**int** k = 0; k < N; k++) {

**if** (found[k]) **continue**;

Relax(s, k, cap[s][k] - flow[s][k], cost[s][k], 1);

Relax(s, k, flow[k][s], -cost[k][s], -1);

**if** (best == -1 || dist[k] < dist[best]) best = k;

}

s = best;

}

**for** (**int** k = 0; k < N; k++)

pi[k] = min(pi[k] + dist[k], INF);

**return** width[t];

}

pair<L, L> GetMaxFlow(**int** s, **int** t) {

L totflow = 0, totcost = 0;

**while** (L amt = Dijkstra(s, t)) {

totflow += amt;

**for** (**int** x = t; x != s; x = dad[x].first) {

**if** (dad[x].second == 1) {

flow[dad[x].first][x] += amt;

totcost += amt \* cost[dad[x].first][x];

} **else** {

flow[x][dad[x].first] -= amt;

totcost -= amt \* cost[x][dad[x].first];

}

}

}

**return** make\_pair(totflow, totcost);

}

};

int main(){

int n, m, s, t;

int a, b;

LL cost, cap;

pair<L, L> result;

cin >> n >> m >> s >> t;

MinCostMaxFlow flow(n); // vertex from 0 -> n-1

while (m--){

cin >> a >> b >> cap >> cost;

flow.AddEdge(a, b, cap, cost);

}

result = flow.GetMaxFlow(s,t);

cout << “Max flow : ” << result.first << endl;

cout << “Min cost : ” << result.second << endl;

return 0;

}

## PushRelabel.cc

*// Adjacency list implementation of FIFO push relabel maximum flow with the gap relabeling*

*// heuristic. This implementation is significantly faster than straight Ford-Fulkerson. It solves*

*// random problems with 10000 vertices and 1000000 edges in a few seconds, though it is possible*

*// to construct test cases that achieve the worst-case.*

*// Running time: O(|V|^3)*

*// INPUT:*

*// - graph, constructed using AddEdge()*

*// - source*

*// - sink*

*// OUTPUT:*

*// - maximum flow value*

*// - To obtain the actual flow values, look at all edges with*

*// capacity > 0 (zero capacity edges are residual edges).*

**typedef** **long** **long** LL;

**struct** Edge {

**int** from, to, cap, flow, index;

Edge(**int** from, **int** to, **int** cap, **int** flow, **int** index) :

from(from), to(to), cap(cap), flow(flow), index(index) {}

};

**struct** PushRelabel {

**int** N;

vector<vector<Edge> > G;

vector<LL> excess;

vector<**int**> dist, active, count;

queue<**int**> Q;

PushRelabel(**int** N) : N(N), G(N), excess(N), dist(N), active(N), count(2\*N) {}

**void** AddEdge(**int** from, **int** to, **int** cap) {

G[from].push\_back(Edge(from, to, cap, 0, G[to].size()));

**if** (from == to) G[from].back().index++;

G[to].push\_back(Edge(to, from, 0, 0, G[from].size() - 1));

}

**void** Enqueue(**int** v) {

**if** (!active[v] && excess[v] > 0) { active[v] = true; Q.push(v); }

}

**void** Push(Edge &e) {

**int** amt = **int**(min(excess[e.from], LL(e.cap - e.flow)));

**if** (dist[e.from] <= dist[e.to] || amt == 0) **return**;

e.flow += amt;

G[e.to][e.index].flow -= amt;

excess[e.to] += amt;

excess[e.from] -= amt;

Enqueue(e.to);

}

**void** Gap(**int** k) {

**for** (**int** v = 0; v < N; v++) {

**if** (dist[v] < k) **continue**;

count[dist[v]]--;

dist[v] = max(dist[v], N+1);

count[dist[v]]++;

Enqueue(v);

}

}

**void** Relabel(**int** v) {

count[dist[v]]--;

dist[v] = 2\*N;

**for** (**int** i = 0; i < G[v].size(); i++)

**if** (G[v][i].cap - G[v][i].flow > 0)

dist[v] = min(dist[v], dist[G[v][i].to] + 1);

count[dist[v]]++;

Enqueue(v);

}

**void** Discharge(**int** v) {

**for** (**int** i = 0; excess[v] > 0 && i < G[v].size(); i++) Push(G[v][i]);

**if** (excess[v] > 0) {

**if** (count[dist[v]] == 1)

Gap(dist[v]);

**else**

Relabel(v);

}

}

LL GetMaxFlow(**int** s, **int** t) {

count[0] = N-1;

count[N] = 1;

dist[s] = N;

active[s] = active[t] = true;

**for** (**int** i = 0; i < G[s].size(); i++) {

excess[s] += G[s][i].cap;

Push(G[s][i]);

}

**while** (!Q.empty()) {

**int** v = Q.front();

Q.pop();

active[v] = false;

Discharge(v);

}

LL totflow = 0;

**for** (**int** i = 0; i < G[s].size(); i++) totflow += G[s][i].flow;

**return** totflow;

}

};

int main(){

int n, m, s, t;

int a, b, c;

cin >> n >> m >> s >> t;

PushRelabel sol(n); // vertex from 0 -> n - 1

while (m--){

cin >> a >> b >> c;

sol.AddEdge(a, b, c);

}

cout << sol.GetMaxFlow(s, t) << endl;

return 0;

}

## MinCut.cc

*// Adjacency matrix implementation of Stoer-Wagner min cut algorithm.*

*// Running time: O(|V|^3)*

*// INPUT: graph, constructed using AddEdge()*

*// OUTPUT:(min cut value, nodes in half of min cut)*

**typedef** vector<**int**> VI;

**typedef** vector<VI> VVI;

**const** **int** INF = 1000000000;

pair<**int**, VI> GetMinCut(VVI &weights) {

**int** N = weights.size();

VI used(N), cut, best\_cut;

**int** best\_weight = -1;

**for** (**int** phase = N-1; phase >= 0; phase--) {

VI w = weights[0];

VI added = used;

**int** prev, last = 0;

**for** (**int** i = 0; i < phase; i++) {

prev = last;

last = -1;

**for** (**int** j = 1; j < N; j++)

**if** (!added[j] && (last == -1 || w[j] > w[last])) last = j;

**if** (i == phase-1) {

**for** (**int** j = 0; j < N; j++) weights[prev][j] += weights[last][j];

**for** (**int** j = 0; j < N; j++) weights[j][prev] = weights[prev][j];

used[last] = true;

cut.push\_back(last);

**if** (best\_weight == -1 || w[last] < best\_weight) {

best\_cut = cut;

best\_weight = w[last];

}

} **else** {

**for** (**int** j = 0; j < N; j++) w[j] += weights[last][j];

added[last] = true;

}

}

}

**return** make\_pair(best\_weight, best\_cut);

}

## MinCostMatching.cc

*// Min cost bipartite matching via shortest augmenting paths. This is an O(n^3) implementation of*

*// a shortest augmenting path algorithm for finding min cost perfect matchings in dense graphs.*

*// In practice, it solves 1000x1000 problems in around 1 second.*

*// cost[i][j] = cost for pairing left node i with right node j*

*// Lmate[i] = index of right node that left node i pairs with*

*// Rmate[j] = index of left node that right node j pairs with*

*// The values in cost[i][j] may be positive or negative. To perform maximization, simply negate*

*// the cost[][] matrix.*

**typedef** vector<**double**> VD;

**typedef** vector<VD> VVD;

**typedef** vector<**int**> VI;

**double** **MinCostMatching**(**const** VVD &cost, VI &Lmate, VI &Rmate) {

**int** n = **int**(cost.size());

*// construct dual feasible solution*

VD u(n);

VD v(n);

**for** (**int** i = 0; i < n; i++) {

u[i] = cost[i][0];

**for** (**int** j = 1; j < n; j++) u[i] = min(u[i], cost[i][j]);

}

**for** (**int** j = 0; j < n; j++) {

v[j] = cost[0][j] - u[0];

**for** (**int** i = 1; i < n; i++) v[j] = min(v[j], cost[i][j] - u[i]);

}

*// construct primal solution satisfying complementary slackness*

Lmate = VI(n, -1);

Rmate = VI(n, -1);

**int** mated = 0;

**for** (**int** i = 0; i < n; i++) {

**for** (**int** j = 0; j < n; j++) {

**if** (Rmate[j] != -1) **continue**;

**if** (fabs(cost[i][j] - u[i] - v[j]) < 1e-10) {

Lmate[i] = j;

Rmate[j] = i;

mated++;

**break**;

}

}

}

VD dist(n);

VI dad(n);

VI seen(n);

*// repeat until primal solution is feasible*

**while** (mated < n) {

*// find an unmatched left node*

**int** s = 0;

**while** (Lmate[s] != -1) s++;

*// initialize Dijkstra*

fill(dad.begin(), dad.end(), -1);

fill(seen.begin(), seen.end(), 0);

**for** (**int** k = 0; k < n; k++)

dist[k] = cost[s][k] - u[s] - v[k];

**int** j = 0;

**while** (true) {

*// find closest*

j = -1;

**for** (**int** k = 0; k < n; k++) {

**if** (seen[k]) **continue**;

**if** (j == -1 || dist[k] < dist[j]) j = k;

}

seen[j] = 1;

*// termination condition*

**if** (Rmate[j] == -1) **break**;

*// relax neighbors*

**const** **int** i = Rmate[j];

**for** (**int** k = 0; k < n; k++) {

**if** (seen[k]) **continue**;

**const** **double** new\_dist = dist[j] + cost[i][k] - u[i] - v[k];

**if** (dist[k] > new\_dist) {

dist[k] = new\_dist;

dad[k] = j;

}

}

}

*// update dual variables*

**for** (**int** k = 0; k < n; k++) {

**if** (k == j || !seen[k]) **continue**;

**const** **int** i = Rmate[k];

v[k] += dist[k] - dist[j];

u[i] -= dist[k] - dist[j];

}

u[s] += dist[j];

*// augment along path*

**while** (dad[j] >= 0) {

**const** **int** d = dad[j];

Rmate[j] = Rmate[d];

Lmate[Rmate[j]] = j;

j = d;

}

Rmate[j] = s;

Lmate[s] = j;

mated++;

}

**double** value = 0;

**for** (**int** i = 0; i < n; i++) value += cost[i][Lmate[i]];

**return** value;

}

* 1. **Segment tree/ Interval Tree**

#define N 20

#define MAX (1+(1<<6)) // Why? :D

#define inf 0x7fffffff

int arr**[**N**];**

int tree**[**MAX**];**

int lazy**[**MAX**];**

/\*\*

\* Build and init tree

\*/

void **build\_tree(**int node**,** int a**,** int b**)** **{**

**if(**a **>** b**)** **return;** // Out of range

**if(**a **==** b**)** **{** // Leaf node

tree**[**node**]** **=** arr**[**a**];** // Init value

**return;**

**}**

build\_tree**(**node**\***2**,** a**,** **(**a**+**b**)/**2**);** // Init left child

build\_tree**(**node**\***2**+**1**,** 1**+(**a**+**b**)/**2**,** b**);** // Init right child

tree**[**node**]** **=** max**(**tree**[**node**\***2**],** tree**[**node**\***2**+**1**]);** // Init root value

**}**

/\*\*

\* Increment elements within range [i, j] with value value

\*/

void **update\_tree(**int node**,** int a**,** int b**,** int i**,** int j**,** int value**)** **{**

**if(**lazy**[**node**]** **!=** 0**)** **{** // This node needs to be updated

tree**[**node**]** **+=** lazy**[**node**];** // Update it

**if(**a **!=** b**)** **{**

lazy**[**node**\***2**]** **+=** lazy**[**node**];** // Mark child as lazy

lazy**[**node**\***2**+**1**]** **+=** lazy**[**node**];** // Mark child as lazy

**}**

lazy**[**node**]** **=** 0**;** // Reset it

**}**

**if(**a **>** b **||** a **>** j **||** b **<** i**)** // Current segment is not within range [i, j]

**return;**

**if(**a **>=** i **&&** b **<=** j**)** **{** // Segment is fully within range

tree**[**node**]** **+=** value**;**

**if(**a **!=** b**)** **{** // Not leaf node

lazy**[**node**\***2**]** **+=** value**;**

lazy**[**node**\***2**+**1**]** **+=** value**;**

**}**

**return;**

**}**

update\_tree**(**node**\***2**,** a**,** **(**a**+**b**)/**2**,** i**,** j**,** value**);** // Updating left child

update\_tree**(**1**+**node**\***2**,** 1**+(**a**+**b**)/**2**,** b**,** i**,** j**,** value**);** // Updating right child

tree**[**node**]** **=** max**(**tree**[**node**\***2**],** tree**[**node**\***2**+**1**]);** // Updating root with max value

**}**

/\*\*

\* Query tree to get max element value within range [i, j]

\*/

int **query\_tree(**int node**,** int a**,** int b**,** int i**,** int j**)** **{**

**if(**a **>** b **||** a **>** j **||** b **<** i**)** **return** **-**inf**;** // Out of range

**if(**lazy**[**node**]** **!=** 0**)** **{** // This node needs to be updated

tree**[**node**]** **+=** lazy**[**node**];** // Update it

**if(**a **!=** b**)** **{**

lazy**[**node**\***2**]** **+=** lazy**[**node**];** // Mark child as lazy

lazy**[**node**\***2**+**1**]** **+=** lazy**[**node**];** // Mark child as lazy

**}**

lazy**[**node**]** **=** 0**;** // Reset it

**}**

**if(**a **>=** i **&&** b **<=** j**)** // Current segment is totally within range [i, j]

**return** tree**[**node**];**

int q1 **=** query\_tree**(**node**\***2**,** a**,** **(**a**+**b**)/**2**,** i**,** j**);** // Query left child

int q2 **=** query\_tree**(**1**+**node**\***2**,** 1**+(**a**+**b**)/**2**,** b**,** i**,** j**);** // Query right child

int res **=** max**(**q1**,** q2**);** // Return final result

**return** res**;**

**}**

int **main()** **{**

**for(**int i **=** 0**;** i **<** N**;** i**++)** arr**[**i**]** **=** 1**;**

build\_tree**(**1**,** 0**,** N**-**1**);**

memset**(**lazy**,** 0**,** **sizeof** lazy**);**

update\_tree**(**1**,** 0**,** N**-**1**,** 0**,** 6**,** 5**);** // Increment range [0, 6] by 5

update\_tree**(**1**,** 0**,** N**-**1**,** 7**,** 10**,** 12**);** // Incremenet range [7, 10] by 12

update\_tree**(**1**,** 0**,** N**-**1**,** 10**,** N**-**1**,** 100**);** // Increment range [10, N-1] by 100

cout **<<** query\_tree**(**1**,** 0**,** N**-**1**,** 0**,** N**-**1**)** **<<** endl**;** // Get max element in range [0, N-1]

**}**

# String Processing

|  |  |
| --- | --- |
| **Convert string <-> number:**  char str[10000];  int a = 123;  sprintf (buffer, "%d", a); // buffer = “123”  buffer[3] = ‘4’;  int b = atoi (buffer); // b = 1234 |  |

* 1. KMP

char text**[**100000**],** pat**[**100**];**

int f**[**100**];**

/\* kmpsetup: setup back tracking links, pat is the pattern, back tracing links are recorded in f. \*/

void kmpsetup**(**char **\***pat**,** int**\*** f**)** **{**

int i**,** k**,** len **=** strlen**(**pat**);**

**for** **(**f**[**0**]** **=** **-**1**,** i **=** 1**;** i **<** len**;** i**++)** **{**

k **=** f**[**i **-** 1**];**

**while** **(**k **>=** 0**)**

**if** **(**pat**[**k**]** **==** pat**[**i **-** 1**])** **break;**

**else** k **=** f**[**k**];**

f**[**i**]** **=** k **+** 1**;**

**}**

**}**

/\* kmpscan: find substring pat in string text using back tracing link recorded in f. \*/

int kmpscan**(**char **\***pat**,** char **\***text**,** int **\***f**)** **{**

int i**,** k**,** ret **=** **-**1**,** len **=** strlen**(**pat**);**

**for** **(**i **=** k **=** 0**;** text**[**i**];)** **{**

**if** **(**k **==** **-**1**)** **{**

i**++;** k **=** 0**;**

**}** **else** **if** **(**text**[**i**]** **==** pat**[**k**])** **{**

i**++;** k**++;**

**if** **(**k **>=** len**)** **{**

ret **=** i **-** len**;**

**break;**

**}**

**}** **else** k **=** f**[**k**];**

**}**

**return** ret**;**

**}**

int main**(**void**)** **{**

int r**;**

scanf**(**"%s %s"**,** text**,** pat**);**

kmpsetup**(**pat**,** f**);**

r **=** kmpscan**(**pat**,** text**,** f**);**

**if** **(**r **==** **-**1**)**

printf**(**"Not Found\n"**);**

**else**

printf**(**"substring starts at %d\n"**,** r**);**

**return** 0**;**

**}**

* 1. Suffix Array

struct SuffixArray

**{**

const int L**;**

string s**;**

vector**<**vector**<**int**>** **>** P**;**

vector**<**pair**<**pair**<**int**,**int**>,** int**>** **>** M**;**

SuffixArray**(**const string **&**s**)** **:** L**(**s**.**length**()),** s**(**s**),** P**(**1**,** vector**<**int**>(**L**,** 0**)),** M**(**L**)**

**{**

**for** **(**int i **=** 0**;** i **<** L**;** i**++)**

P**[**0**][**i**]** **=** int**(**s**[**i**]);**

**for** **(**int skip **=** 1**,** level **=** 1**;** skip **<** L**;** skip **\*=** 2**,** level**++)**

**{**

P**.**push\_back**(**vector**<**int**>(**L**,** 0**));**

**for** **(**int i **=** 0**;** i **<** L**;** i**++)**

M**[**i**]** **=** make\_pair**(**make\_pair**(**P**[**level **-** 1**][**i**],** i **+** skip **<** L **?** P**[**level**-**1**][**i**+**skip**]:-**1000**),**

i**);**

sort**(**M**.**begin**(),** M**.**end**());**

**for** **(**int i **=** 0**;** i **<** L**;** i**++)**

P**[**level**][**M**[**i**].**second**]** **=** **(**i**>**0 **&&** M**[**i**].**first **==** M**[**i**-**1**].**first**)** **?** P**[**level**][**M**[**i**-**1**].**second**]:**

i**;**

**}**

**}**

vector**<**int**>** GetSuffixArray**()** **{** **return** P**.**back**();}**

int LongestCommonPrefix**(**int i**,** int j**)**

**{**

int len **=** 0**;**

**if** **(**i **==** j**)** **return** L **-** i**;**

**for** **(**int k **=** P**.**size**()** **-** 1**;** k **>=** 0 **&&** i **<** L **&&** j **<** L**;** k**--)**

**if** **(**P**[**k**][**i**]** **==** P**[**k**][**j**])**

**{**

i **+=** 1 **<<** k**;**

j **+=** 1 **<<** k**;**

len **+=** 1 **<<**k**;**

**}**

**return** len**;**

**}**

**};**

int main**()**

**{**

SuffixArray suffix**(**"bobocel"**);**

vector**<**int**>** v **=** suffix**.**GetSuffixArray**();**

**for** **(**int i **=** 0**;** i **<** v**.**size**();** i**++)**

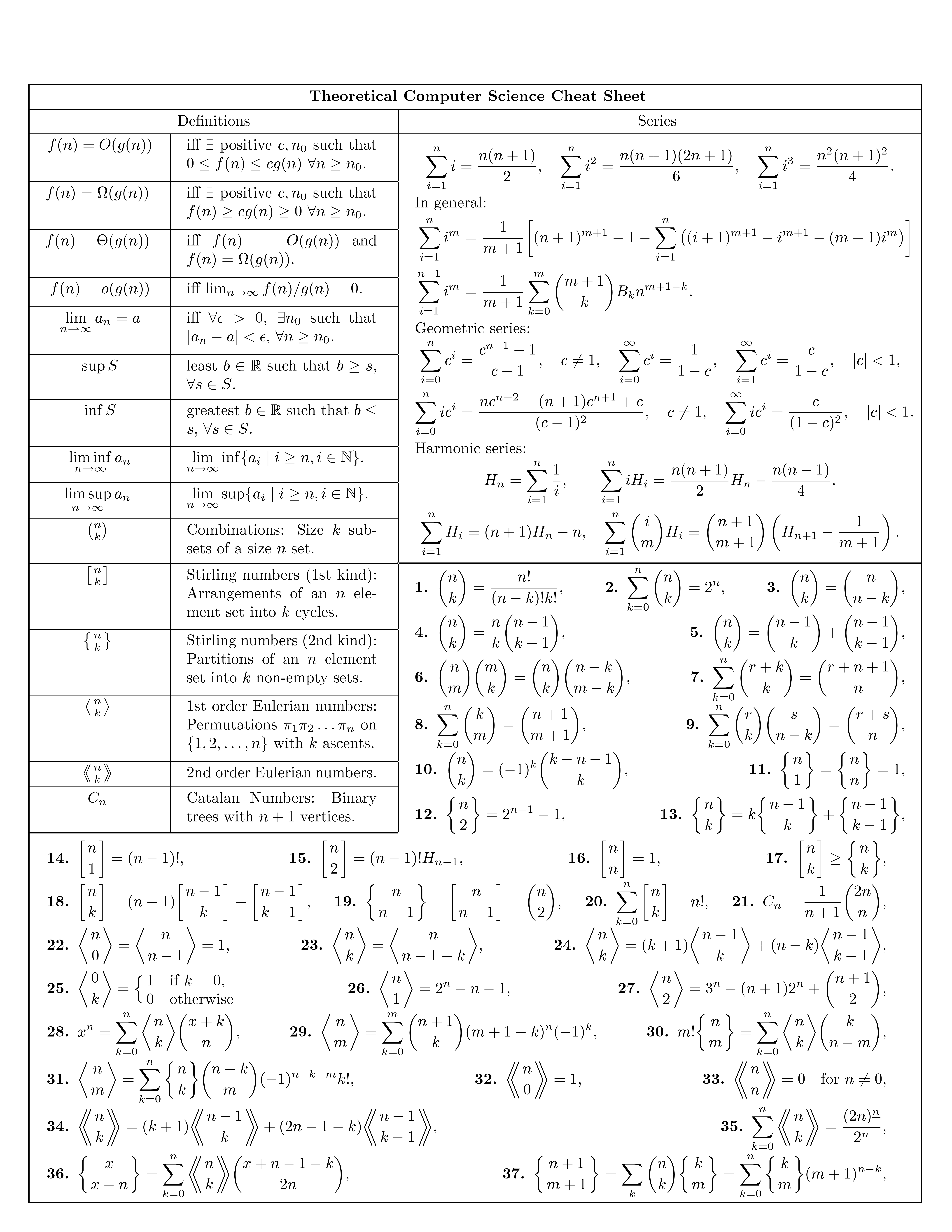
cout**<<**v**[**i**]<<**endl**;**

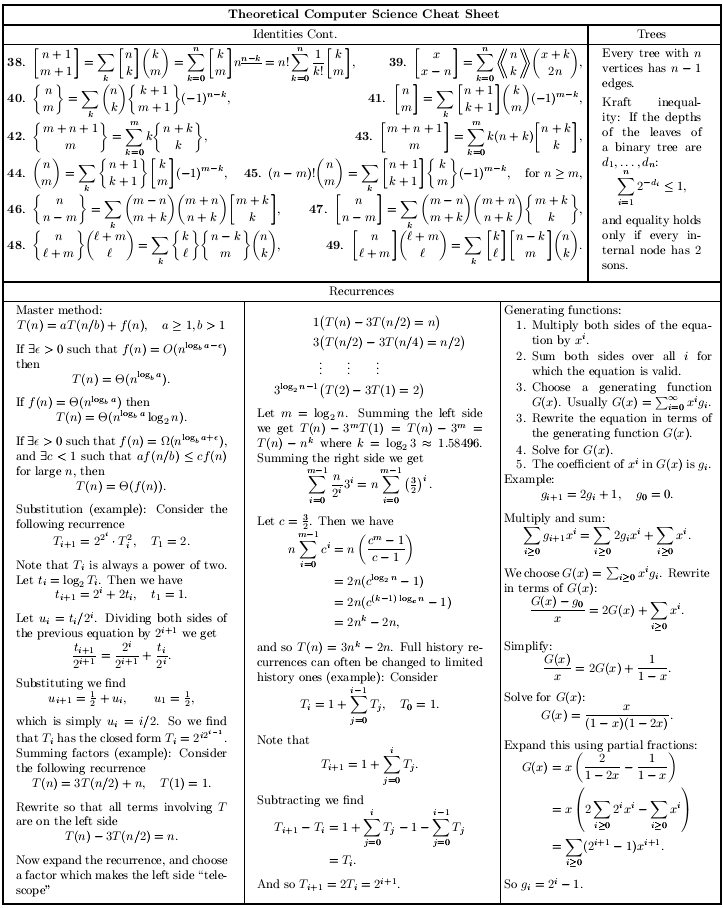
cout**<<**suffix**.**LongestCommonPrefix**(**0**,** 2**)<<**endl**;**

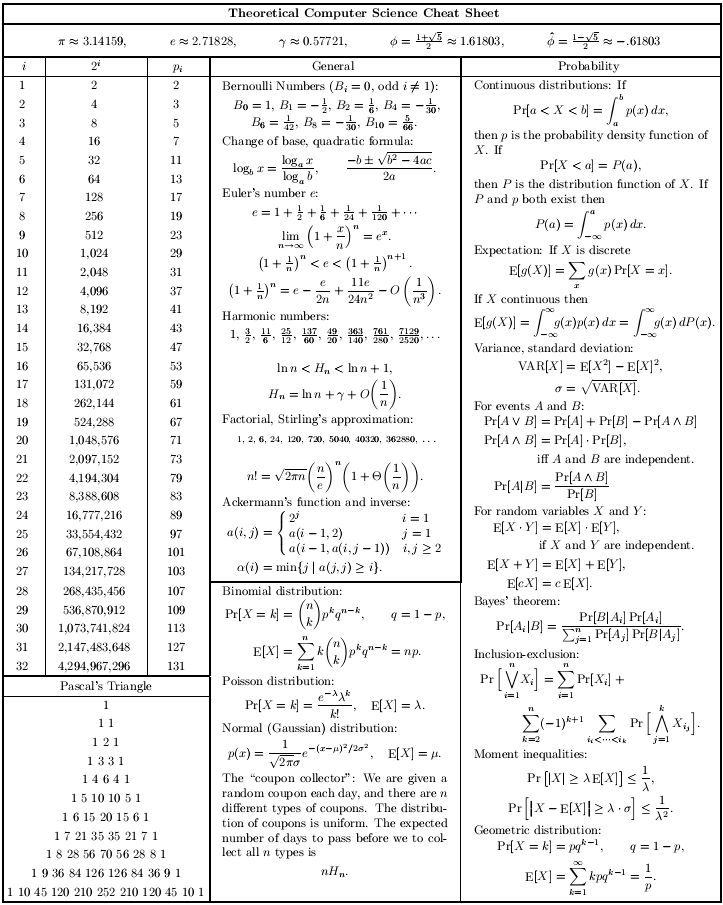
**return** 0**;**

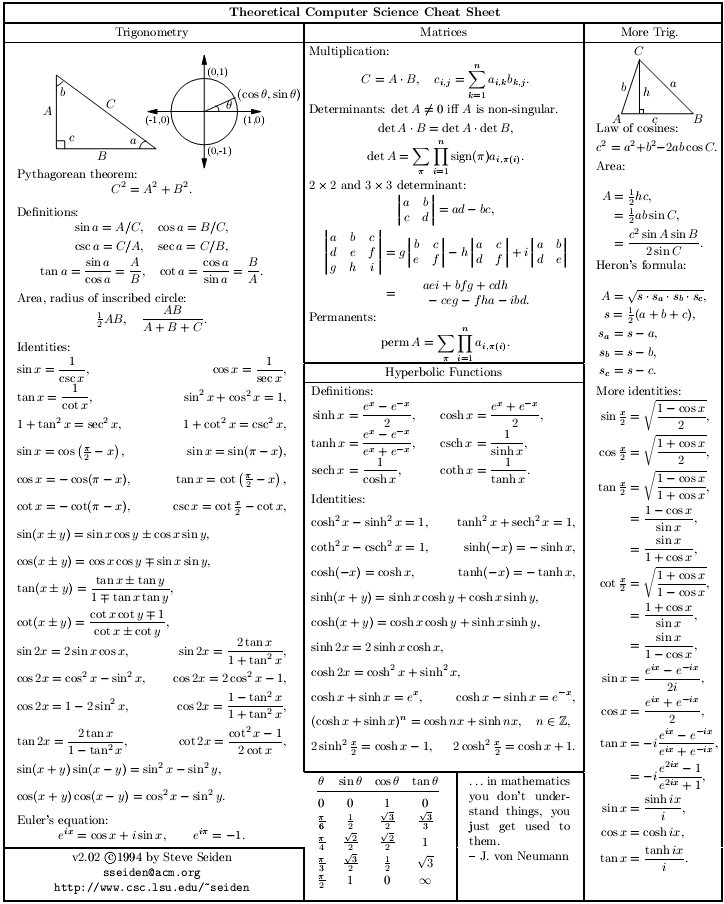
**}**

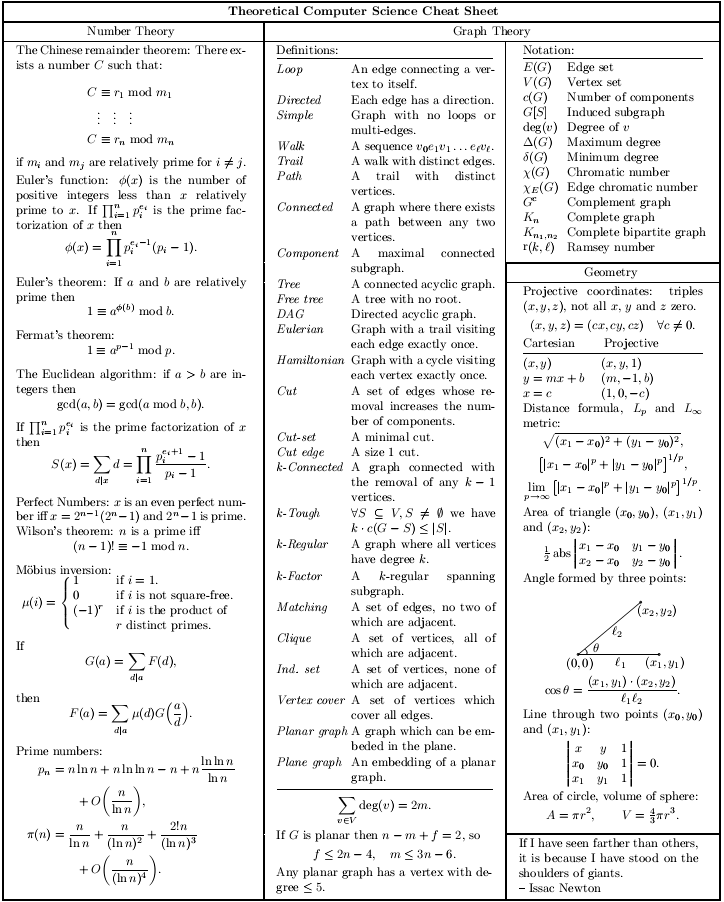
# Appendix











**Legend**:

|  |  |
| --- | --- |
| **C++98** | Available since C++98 |
| **C++11** | New in C++11 |

#### Sequence containers

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Headers | | [**<array>**](http://www.cplusplus.com/%3Carray%3E) | [**<vector>**](http://www.cplusplus.com/%3Cvector%3E) | [**<deque>**](http://www.cplusplus.com/%3Cdeque%3E) | [**<forward\_list>**](http://www.cplusplus.com/%3Cforward_list%3E) | [**<list>**](http://www.cplusplus.com/%3Clist%3E) |
| Members | | [**array**](http://www.cplusplus.com/array) | [**vector**](http://www.cplusplus.com/vector) | [**deque**](http://www.cplusplus.com/deque) | [**forward\_list**](http://www.cplusplus.com/forward_list) | [**list**](http://www.cplusplus.com/list) |
|  | *constructor* | *implicit* | [vector](http://www.cplusplus.com/vector::vector) | [deque](http://www.cplusplus.com/deque::deque) | [forward\_list](http://www.cplusplus.com/forward_list::forward_list) | [list](http://www.cplusplus.com/list::list) |
| *destructor* | *implicit* | [~vector](http://www.cplusplus.com/vector::~vector) | [~deque](http://www.cplusplus.com/deque::~deque) | [~forward\_list](http://www.cplusplus.com/forward_list::~forward_list) | [~list](http://www.cplusplus.com/list::~list) |
| operator= | *implicit* | [operator=](http://www.cplusplus.com/vector::operator=) | [operator=](http://www.cplusplus.com/deque::operator=) | [operator=](http://www.cplusplus.com/forward_list::operator=) | [operator=](http://www.cplusplus.com/list::operator=) |
| iterators | begin | [begin](http://www.cplusplus.com/array::begin) | [begin](http://www.cplusplus.com/vector::begin) | [begin](http://www.cplusplus.com/deque::begin) | [begin](http://www.cplusplus.com/forward_list::begin) [before\_begin](http://www.cplusplus.com/forward_list::before_begin) | [begin](http://www.cplusplus.com/list::begin) |
| end | [end](http://www.cplusplus.com/array::end) | [end](http://www.cplusplus.com/vector::end) | [end](http://www.cplusplus.com/deque::end) | [end](http://www.cplusplus.com/forward_list::end) | [end](http://www.cplusplus.com/list::end) |
| rbegin | [rbegin](http://www.cplusplus.com/array::rbegin) | [rbegin](http://www.cplusplus.com/vector::rbegin) | [rbegin](http://www.cplusplus.com/deque::rbegin) |  | [rbegin](http://www.cplusplus.com/list::rbegin) |
| rend | [rend](http://www.cplusplus.com/array::rend) | [rend](http://www.cplusplus.com/vector::rend) | [rend](http://www.cplusplus.com/deque::rend) |  | [rend](http://www.cplusplus.com/list::rend) |
| const iterators | begin | [cbegin](http://www.cplusplus.com/array::cbegin) | [cbegin](http://www.cplusplus.com/vector::cbegin) | [cbegin](http://www.cplusplus.com/deque::cbegin) | [cbegin](http://www.cplusplus.com/forward_list::cbegin) [cbefore\_begin](http://www.cplusplus.com/forward_list::cbefore_begin) | [cbegin](http://www.cplusplus.com/list::cbegin) |
| cend | [cend](http://www.cplusplus.com/array::cend) | [cend](http://www.cplusplus.com/vector::cend) | [cend](http://www.cplusplus.com/deque::cend) | [cend](http://www.cplusplus.com/forward_list::cend) | [cend](http://www.cplusplus.com/list::cend) |
| crbegin | [crbegin](http://www.cplusplus.com/array::crbegin) | [crbegin](http://www.cplusplus.com/vector::crbegin) | [crbegin](http://www.cplusplus.com/deque::crbegin) |  | [crbegin](http://www.cplusplus.com/list::crbegin) |
| crend | [crend](http://www.cplusplus.com/array::crend) | [crend](http://www.cplusplus.com/vector::crend) | [crend](http://www.cplusplus.com/deque::crend) |  | [crend](http://www.cplusplus.com/list::crend) |
| capacity | size | [size](http://www.cplusplus.com/array::size) | [size](http://www.cplusplus.com/vector::size) | [size](http://www.cplusplus.com/deque::size) |  | [size](http://www.cplusplus.com/list::size) |
| max\_size | [max\_size](http://www.cplusplus.com/array::max_size) | [max\_size](http://www.cplusplus.com/vector::max_size) | [max\_size](http://www.cplusplus.com/deque::max_size) | [max\_size](http://www.cplusplus.com/forward_list::max_size) | [max\_size](http://www.cplusplus.com/list::max_size) |
| empty | [empty](http://www.cplusplus.com/array::empty) | [empty](http://www.cplusplus.com/vector::empty) | [empty](http://www.cplusplus.com/deque::empty) | [empty](http://www.cplusplus.com/forward_list::empty) | [empty](http://www.cplusplus.com/list::empty) |
| resize |  | [resize](http://www.cplusplus.com/vector::resize) | [resize](http://www.cplusplus.com/deque::resize) | [resize](http://www.cplusplus.com/forward_list::resize) | [resize](http://www.cplusplus.com/list::resize) |
| shrink\_to\_fit |  | [shrink\_to\_fit](http://www.cplusplus.com/vector::shrink_to_fit) | [shrink\_to\_fit](http://www.cplusplus.com/deque::shrink_to_fit) |  |  |
| capacity |  | [capacity](http://www.cplusplus.com/vector::capacity) |  |  |  |
| reserve |  | [reserve](http://www.cplusplus.com/vector::reserve) |  |  |  |
| element access | front | [front](http://www.cplusplus.com/array::front) | [front](http://www.cplusplus.com/vector::front) | [front](http://www.cplusplus.com/deque::front) | [front](http://www.cplusplus.com/forward_list::front) | [front](http://www.cplusplus.com/list::front) |
| back | [back](http://www.cplusplus.com/array::back) | [back](http://www.cplusplus.com/vector::back) | [back](http://www.cplusplus.com/deque::back) |  | [back](http://www.cplusplus.com/list::back) |
| operator[] | [operator[]](http://www.cplusplus.com/array::operator%5b%5d) | [operator[]](http://www.cplusplus.com/vector::operator%5b%5d) | [operator[]](http://www.cplusplus.com/deque::operator%5b%5d) |  |  |
| at | [at](http://www.cplusplus.com/array::at) | [at](http://www.cplusplus.com/vector::at) | [at](http://www.cplusplus.com/deque::at) |  |  |
| modifiers | assign |  | [assign](http://www.cplusplus.com/vector::assign) | [assign](http://www.cplusplus.com/deque::assign) | [assign](http://www.cplusplus.com/forward_list::assign) | [assign](http://www.cplusplus.com/list::assign) |
| emplace |  | [emplace](http://www.cplusplus.com/vector::emplace) | [emplace](http://www.cplusplus.com/deque::emplace) | [emplace\_after](http://www.cplusplus.com/forward_list::emplace_after) | [emplace](http://www.cplusplus.com/list::emplace) |
| insert |  | [insert](http://www.cplusplus.com/vector::insert) | [insert](http://www.cplusplus.com/deque::insert) | [insert\_after](http://www.cplusplus.com/forward_list::insert_after) | [insert](http://www.cplusplus.com/list::insert) |
| erase |  | [erase](http://www.cplusplus.com/vector::erase) | [erase](http://www.cplusplus.com/deque::erase) | [erase\_after](http://www.cplusplus.com/forward_list::erase_after) | [erase](http://www.cplusplus.com/list::erase) |
| emplace\_back |  | [emplace\_back](http://www.cplusplus.com/vector::emplace_back) | [emplace\_back](http://www.cplusplus.com/deque::emplace_back) |  | [emplace\_back](http://www.cplusplus.com/list::emplace_back) |
| push\_back |  | [push\_back](http://www.cplusplus.com/vector::push_back) | [push\_back](http://www.cplusplus.com/deque::push_back) |  | [push\_back](http://www.cplusplus.com/list::push_back) |
| pop\_back |  | [pop\_back](http://www.cplusplus.com/vector::pop_back) | [pop\_back](http://www.cplusplus.com/deque::pop_back) |  | [pop\_back](http://www.cplusplus.com/list::pop_back) |
| emplace\_front |  |  | [emplace\_front](http://www.cplusplus.com/deque::emplace_front) | [emplace\_front](http://www.cplusplus.com/forward_list::emplace_front) | [emplace\_front](http://www.cplusplus.com/list::emplace_front) |
| push\_front |  |  | [push\_front](http://www.cplusplus.com/deque::push_front) | [push\_front](http://www.cplusplus.com/forward_list::push_front) | [push\_front](http://www.cplusplus.com/list::push_front) |
| pop\_front |  |  | [pop\_front](http://www.cplusplus.com/deque::pop_front) | [pop\_front](http://www.cplusplus.com/forward_list::pop_front) | [pop\_front](http://www.cplusplus.com/list::pop_front) |
| clear |  | [clear](http://www.cplusplus.com/vector::clear) | [clear](http://www.cplusplus.com/deque::clear) | [clear](http://www.cplusplus.com/forward_list::clear) | [clear](http://www.cplusplus.com/list::clear) |
| swap | [swap](http://www.cplusplus.com/array::swap) | [swap](http://www.cplusplus.com/vector::swap) | [swap](http://www.cplusplus.com/deque::swap) | [swap](http://www.cplusplus.com/forward_list::swap) | [swap](http://www.cplusplus.com/list::swap) |
| list operations | splice |  |  |  | [splice\_after](http://www.cplusplus.com/forward_list::splice_after) | [splice](http://www.cplusplus.com/list::splice) |
| remove |  |  |  | [remove](http://www.cplusplus.com/forward_list::remove) | [remove](http://www.cplusplus.com/list::remove) |
| remove\_if |  |  |  | [remove\_if](http://www.cplusplus.com/forward_list::remove_if) | [remove\_if](http://www.cplusplus.com/list::remove_if) |
| unique |  |  |  | [unique](http://www.cplusplus.com/forward_list::unique) | [unique](http://www.cplusplus.com/list::unique) |
| merge |  |  |  | [merge](http://www.cplusplus.com/forward_list::merge) | [merge](http://www.cplusplus.com/list::merge) |
| sort |  |  |  | [sort](http://www.cplusplus.com/forward_list::sort) | [sort](http://www.cplusplus.com/list::sort) |
| reverse |  |  |  | [reverse](http://www.cplusplus.com/forward_list::reverse) | [reverse](http://www.cplusplus.com/list::reverse) |
| observers | get\_allocator |  | [get\_allocator](http://www.cplusplus.com/vector::get_allocator) | [get\_allocator](http://www.cplusplus.com/deque::get_allocator) | [get\_allocator](http://www.cplusplus.com/forward_list::get_allocator) | [get\_allocator](http://www.cplusplus.com/list::get_allocator) |
| data | [data](http://www.cplusplus.com/array::data) | [data](http://www.cplusplus.com/vector::data) |  |  |  |

#### Associative containers

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Headers | | [**<set>**](http://www.cplusplus.com/%3Cset%3E) | | [**<map>**](http://www.cplusplus.com/%3Cmap%3E) | | [**<unordered\_set>**](http://www.cplusplus.com/%3Cunordered_set%3E) | | [**<unordered\_map>**](http://www.cplusplus.com/%3Cunordered_map%3E) | |
| Members | | [**set**](http://www.cplusplus.com/set) | [**multiset**](http://www.cplusplus.com/multiset) | [**map**](http://www.cplusplus.com/map) | [**multimap**](http://www.cplusplus.com/multimap) | [**unordered\_set**](http://www.cplusplus.com/unordered_set) | [**unordered\_multiset**](http://www.cplusplus.com/unordered_multiset) | [**unordered\_map**](http://www.cplusplus.com/unordered_map) | [**unordered\_multimap**](http://www.cplusplus.com/unordered_multimap) |
|  | *constructor* | [set](http://www.cplusplus.com/set::set) | [multiset](http://www.cplusplus.com/multiset::multiset) | [map](http://www.cplusplus.com/map::map) | [multimap](http://www.cplusplus.com/multimap::multimap) | [unordered\_set](http://www.cplusplus.com/unordered_set::unordered_set) | [unordered\_multiset](http://www.cplusplus.com/unordered_multiset::unordered_multiset) | [unordered\_map](http://www.cplusplus.com/unordered_map::unordered_map) | [unordered\_multimap](http://www.cplusplus.com/unordered_multimap::unordered_multimap) |
| *destructor* | [~set](http://www.cplusplus.com/set::~set) | [~multiset](http://www.cplusplus.com/multiset::~multiset) | [~map](http://www.cplusplus.com/map::~map) | [~multimap](http://www.cplusplus.com/multimap::~multimap) | [~unordered\_set](http://www.cplusplus.com/unordered_set::~unordered_set) | [~unordered\_multiset](http://www.cplusplus.com/unordered_multiset::~unordered_multiset) | [~unordered\_map](http://www.cplusplus.com/unordered_map::~unordered_map) | [~unordered\_multimap](http://www.cplusplus.com/unordered_multimap::~unordered_multimap) |
| *assignment* | [operator=](http://www.cplusplus.com/set::operator=) | [operator=](http://www.cplusplus.com/multiset::operator=) | [operator=](http://www.cplusplus.com/map::operator=) | [operator=](http://www.cplusplus.com/multimap::operator=) | [operator=](http://www.cplusplus.com/unordered_set::operator=) | [operator=](http://www.cplusplus.com/unordered_multiset::operator=) | [operator=](http://www.cplusplus.com/unordered_map::operator=) | [operator=](http://www.cplusplus.com/unordered_multimap::operator=) |
| iterators | begin | [begin](http://www.cplusplus.com/set::begin) | [begin](http://www.cplusplus.com/multiset::begin) | [begin](http://www.cplusplus.com/map::begin) | [begin](http://www.cplusplus.com/multimap::begin) | [begin](http://www.cplusplus.com/unordered_set::begin) | [begin](http://www.cplusplus.com/unordered_multiset::begin) | [begin](http://www.cplusplus.com/unordered_map::begin) | [begin](http://www.cplusplus.com/unordered_multimap::begin) |
| end | [end](http://www.cplusplus.com/set::end) | [end](http://www.cplusplus.com/multiset::end) | [end](http://www.cplusplus.com/map::end) | [end](http://www.cplusplus.com/multimap::end) | [end](http://www.cplusplus.com/unordered_set::end) | [end](http://www.cplusplus.com/unordered_multiset::end) | [end](http://www.cplusplus.com/unordered_map::end) | [end](http://www.cplusplus.com/unordered_multimap::end) |
| rbegin | [rbegin](http://www.cplusplus.com/set::rbegin) | [rbegin](http://www.cplusplus.com/multiset::rbegin) | [rbegin](http://www.cplusplus.com/map::rbegin) | [rbegin](http://www.cplusplus.com/multimap::rbegin) |  |  |  |  |
| rend | [rend](http://www.cplusplus.com/set::rend) | [rend](http://www.cplusplus.com/multiset::rend) | [rend](http://www.cplusplus.com/map::rend) | [rend](http://www.cplusplus.com/multimap::rend) |  |  |  |  |
| const iterators | cbegin | [cbegin](http://www.cplusplus.com/set::cbegin) | [cbegin](http://www.cplusplus.com/multiset::cbegin) | [cbegin](http://www.cplusplus.com/map::cbegin) | [cbegin](http://www.cplusplus.com/multimap::cbegin) | [cbegin](http://www.cplusplus.com/unordered_set::cbegin) | [cbegin](http://www.cplusplus.com/unordered_multiset::cbegin) | [cbegin](http://www.cplusplus.com/unordered_map::cbegin) | [cbegin](http://www.cplusplus.com/unordered_multimap::cbegin) |
| cend | [cend](http://www.cplusplus.com/set::cend) | [cend](http://www.cplusplus.com/multiset::cend) | [cend](http://www.cplusplus.com/map::cend) | [cend](http://www.cplusplus.com/multimap::cend) | [cend](http://www.cplusplus.com/unordered_set::cend) | [cend](http://www.cplusplus.com/unordered_multiset::cend) | [cend](http://www.cplusplus.com/unordered_map::cend) | [cend](http://www.cplusplus.com/unordered_multimap::cend) |
| crbegin | [crbegin](http://www.cplusplus.com/set::crbegin) | [crbegin](http://www.cplusplus.com/multiset::crbegin) | [crbegin](http://www.cplusplus.com/map::crbegin) | [crbegin](http://www.cplusplus.com/multimap::crbegin) |  |  |  |  |
| crend | [crend](http://www.cplusplus.com/set::crend) | [crend](http://www.cplusplus.com/multiset::crend) | [crend](http://www.cplusplus.com/map::crend) | [crend](http://www.cplusplus.com/multimap::crend) |  |  |  |  |
| capacity | size | [size](http://www.cplusplus.com/set::size) | [size](http://www.cplusplus.com/multiset::size) | [size](http://www.cplusplus.com/map::size) | [size](http://www.cplusplus.com/multimap::size) | [size](http://www.cplusplus.com/unordered_set::size) | [size](http://www.cplusplus.com/unordered_multiset::size) | [size](http://www.cplusplus.com/unordered_map::size) | [size](http://www.cplusplus.com/unordered_multimap::size) |
| max\_size | [max\_size](http://www.cplusplus.com/set::max_size) | [max\_size](http://www.cplusplus.com/multiset::max_size) | [max\_size](http://www.cplusplus.com/map::max_size) | [max\_size](http://www.cplusplus.com/multimap::max_size) | [max\_size](http://www.cplusplus.com/unordered_set::max_size) | [max\_size](http://www.cplusplus.com/unordered_multiset::max_size) | [max\_size](http://www.cplusplus.com/unordered_map::max_size) | [max\_size](http://www.cplusplus.com/unordered_multimap::max_size) |
| empty | [empty](http://www.cplusplus.com/set::empty) | [empty](http://www.cplusplus.com/multiset::empty) | [empty](http://www.cplusplus.com/map::empty) | [empty](http://www.cplusplus.com/multimap::empty) | [empty](http://www.cplusplus.com/unordered_set::empty) | [empty](http://www.cplusplus.com/unordered_multiset::empty) | [empty](http://www.cplusplus.com/unordered_map::empty) | [empty](http://www.cplusplus.com/unordered_multimap::empty) |
| reserve |  |  |  |  | [reserve](http://www.cplusplus.com/unordered_set::reserve) | [reserve](http://www.cplusplus.com/unordered_multiset::reserve) | [reserve](http://www.cplusplus.com/unordered_map::reserve) | [reserve](http://www.cplusplus.com/unordered_multimap::reserve) |
| element access | at |  |  | [at](http://www.cplusplus.com/map::at) |  |  |  | [at](http://www.cplusplus.com/unordered_map::at) |  |
| operator[] |  |  | [operator[]](http://www.cplusplus.com/map::operator%5b%5d) |  |  |  | [operator[]](http://www.cplusplus.com/unordered_map::operator%5b%5d) |  |
| modifiers | emplace | [emplace](http://www.cplusplus.com/set::emplace) | [emplace](http://www.cplusplus.com/multiset::emplace) | [emplace](http://www.cplusplus.com/map::emplace) | [emplace](http://www.cplusplus.com/multimap::emplace) | [emplace](http://www.cplusplus.com/unordered_set::emplace) | [emplace](http://www.cplusplus.com/unordered_multiset::emplace) | [emplace](http://www.cplusplus.com/unordered_map::emplace) | [emplace](http://www.cplusplus.com/unordered_multimap::emplace) |
| emplace\_hint | [emplace\_hint](http://www.cplusplus.com/set::emplace_hint) | [emplace\_hint](http://www.cplusplus.com/multiset::emplace_hint) | [emplace\_hint](http://www.cplusplus.com/map::emplace_hint) | [emplace\_hint](http://www.cplusplus.com/multimap::emplace_hint) | [emplace\_hint](http://www.cplusplus.com/unordered_set::emplace_hint) | [emplace\_hint](http://www.cplusplus.com/unordered_multiset::emplace_hint) | [emplace\_hint](http://www.cplusplus.com/unordered_map::emplace_hint) | [emplace\_hint](http://www.cplusplus.com/unordered_multimap::emplace_hint) |
| insert | [insert](http://www.cplusplus.com/set::insert) | [insert](http://www.cplusplus.com/multiset::insert) | [insert](http://www.cplusplus.com/map::insert) | [insert](http://www.cplusplus.com/multimap::insert) | [insert](http://www.cplusplus.com/unordered_set::insert) | [insert](http://www.cplusplus.com/unordered_multiset::insert) | [insert](http://www.cplusplus.com/unordered_map::insert) | [insert](http://www.cplusplus.com/unordered_multimap::insert) |
| erase | [erase](http://www.cplusplus.com/set::erase) | [erase](http://www.cplusplus.com/multiset::erase) | [erase](http://www.cplusplus.com/map::erase) | [erase](http://www.cplusplus.com/multimap::erase) | [erase](http://www.cplusplus.com/unordered_set::erase) | [erase](http://www.cplusplus.com/unordered_multiset::erase) | [erase](http://www.cplusplus.com/unordered_map::erase) | [erase](http://www.cplusplus.com/unordered_multimap::erase) |
| clear | [clear](http://www.cplusplus.com/set::clear) | [clear](http://www.cplusplus.com/multiset::clear) | [clear](http://www.cplusplus.com/map::clear) | [clear](http://www.cplusplus.com/multimap::clear) | [clear](http://www.cplusplus.com/unordered_set::clear) | [clear](http://www.cplusplus.com/unordered_multiset::clear) | [clear](http://www.cplusplus.com/unordered_map::clear) | [clear](http://www.cplusplus.com/unordered_multimap::clear) |
| swap | [swap](http://www.cplusplus.com/set::swap) | [swap](http://www.cplusplus.com/multiset::swap) | [swap](http://www.cplusplus.com/map::swap) | [swap](http://www.cplusplus.com/multimap::swap) | [swap](http://www.cplusplus.com/unordered_set::swap) | [swap](http://www.cplusplus.com/unordered_multiset::swap) | [swap](http://www.cplusplus.com/unordered_map::swap) | [swap](http://www.cplusplus.com/unordered_multimap::swap) |
| operations | count | [count](http://www.cplusplus.com/set::count) | [count](http://www.cplusplus.com/multiset::count) | [count](http://www.cplusplus.com/map::count) | [count](http://www.cplusplus.com/multimap::count) | [count](http://www.cplusplus.com/unordered_set::count) | [count](http://www.cplusplus.com/unordered_multiset::count) | [count](http://www.cplusplus.com/unordered_map::count) | [count](http://www.cplusplus.com/unordered_multimap::count) |
| find | [find](http://www.cplusplus.com/set::find) | [find](http://www.cplusplus.com/multiset::find) | [find](http://www.cplusplus.com/map::find) | [find](http://www.cplusplus.com/multimap::find) | [find](http://www.cplusplus.com/unordered_set::find) | [find](http://www.cplusplus.com/unordered_multiset::find) | [find](http://www.cplusplus.com/unordered_map::find) | [find](http://www.cplusplus.com/unordered_multimap::find) |
| equal\_range | [equal\_range](http://www.cplusplus.com/set::equal_range) | [equal\_range](http://www.cplusplus.com/multiset::equal_range) | [equal\_range](http://www.cplusplus.com/map::equal_range) | [equal\_range](http://www.cplusplus.com/multimap::equal_range) | [equal\_range](http://www.cplusplus.com/unordered_set::equal_range) | [equal\_range](http://www.cplusplus.com/unordered_multiset::equal_range) | [equal\_range](http://www.cplusplus.com/unordered_map::equal_range) | [equal\_range](http://www.cplusplus.com/unordered_multimap::equal_range) |
| lower\_bound | [lower\_bound](http://www.cplusplus.com/set::lower_bound) | [lower\_bound](http://www.cplusplus.com/multiset::lower_bound) | [lower\_bound](http://www.cplusplus.com/map::lower_bound) | [lower\_bound](http://www.cplusplus.com/multimap::lower_bound) |  |  |  |  |
| upper\_bound | [upper\_bound](http://www.cplusplus.com/set::upper_bound) | [upper\_bound](http://www.cplusplus.com/multiset::upper_bound) | [upper\_bound](http://www.cplusplus.com/map::upper_bound) | [upper\_bound](http://www.cplusplus.com/multimap::upper_bound) |  |  |  |  |
| observers | get\_allocator | [get\_allocator](http://www.cplusplus.com/set::get_allocator) | [get\_allocator](http://www.cplusplus.com/multiset::get_allocator) | [get\_allocator](http://www.cplusplus.com/map::get_allocator) | [get\_allocator](http://www.cplusplus.com/multimap::get_allocator) | [get\_allocator](http://www.cplusplus.com/unordered_set::get_allocator) | [get\_allocator](http://www.cplusplus.com/unordered_multiset::get_allocator) | [get\_allocator](http://www.cplusplus.com/unordered_map::get_allocator) | [get\_allocator](http://www.cplusplus.com/unordered_multimap::get_allocator) |
| key\_comp | [key\_comp](http://www.cplusplus.com/set::key_comp) | [key\_comp](http://www.cplusplus.com/multiset::key_comp) | [key\_comp](http://www.cplusplus.com/map::key_comp) | [key\_comp](http://www.cplusplus.com/multimap::key_comp) |  |  |  |  |
| value\_comp | [value\_comp](http://www.cplusplus.com/set::value_comp) | [value\_comp](http://www.cplusplus.com/multiset::value_comp) | [value\_comp](http://www.cplusplus.com/map::value_comp) | [value\_comp](http://www.cplusplus.com/multimap::value_comp) |  |  |  |  |
| key\_eq |  |  |  |  | [key\_eq](http://www.cplusplus.com/unordered_set::key_eq) | [key\_eq](http://www.cplusplus.com/unordered_multiset::key_eq) | [key\_eq](http://www.cplusplus.com/unordered_map::key_eq) | [key\_eq](http://www.cplusplus.com/unordered_multimap::key_eq) |
| hash\_function |  |  |  |  | [hash\_function](http://www.cplusplus.com/unordered_set::hash_function) | [hash\_function](http://www.cplusplus.com/unordered_multiset::hash_function) | [hash\_function](http://www.cplusplus.com/unordered_map::hash_function) | [hash\_function](http://www.cplusplus.com/unordered_multimap::hash_function) |
| buckets | bucket |  |  |  |  | [bucket](http://www.cplusplus.com/unordered_set::bucket) | [bucket](http://www.cplusplus.com/unordered_multiset::bucket) | [bucket](http://www.cplusplus.com/unordered_map::bucket) | [bucket](http://www.cplusplus.com/unordered_multimap::bucket) |
| bucket\_count |  |  |  |  | [bucket\_count](http://www.cplusplus.com/unordered_set::bucket_count) | [bucket\_count](http://www.cplusplus.com/unordered_multiset::bucket_count) | [bucket\_count](http://www.cplusplus.com/unordered_map::bucket_count) | [bucket\_count](http://www.cplusplus.com/unordered_multimap::bucket_count) |
| bucket\_size |  |  |  |  | [bucket\_size](http://www.cplusplus.com/unordered_set::bucket_size) | [bucket\_size](http://www.cplusplus.com/unordered_multiset::bucket_size) | [bucket\_size](http://www.cplusplus.com/unordered_map::bucket_size) | [bucket\_size](http://www.cplusplus.com/unordered_multimap::bucket_size) |
| max\_bucket\_count |  |  |  |  | [max\_bucket\_count](http://www.cplusplus.com/unordered_set::max_bucket_count) | [max\_bucket\_count](http://www.cplusplus.com/unordered_multiset::max_bucket_count) | [max\_bucket\_count](http://www.cplusplus.com/unordered_map::max_bucket_count) | [max\_bucket\_count](http://www.cplusplus.com/unordered_multimap::max_bucket_count) |
| hash policy | rehash |  |  |  |  | [rehash](http://www.cplusplus.com/unordered_set::rehash) | [rehash](http://www.cplusplus.com/unordered_multiset::rehash) | [rehash](http://www.cplusplus.com/unordered_map::rehash) | [rehash](http://www.cplusplus.com/unordered_multimap::rehash) |
| load\_factor |  |  |  |  | [load\_factor](http://www.cplusplus.com/unordered_set::load_factor) | [load\_factor](http://www.cplusplus.com/unordered_multiset::load_factor) | [load\_factor](http://www.cplusplus.com/unordered_map::load_factor) | [load\_factor](http://www.cplusplus.com/unordered_multimap::load_factor) |
| max\_load\_factor |  |  |  |  | [max\_load\_factor](http://www.cplusplus.com/unordered_set::max_load_factor) | [max\_load\_factor](http://www.cplusplus.com/unordered_multiset::max_load_factor) | [max\_load\_factor](http://www.cplusplus.com/unordered_map::max_load_factor) | [max\_load\_factor](http://www.cplusplus.com/unordered_multimap::max_load_factor) |