Algorytmy i struktury danych
wykład 3
Liczby szczególne: współczynniki dwumianowe, liczby Stirlinga drugiego rodzaju, liczby Bella

Krzysztof Diks
Instytut Informatyki UW

Współczynniki dwumianowe

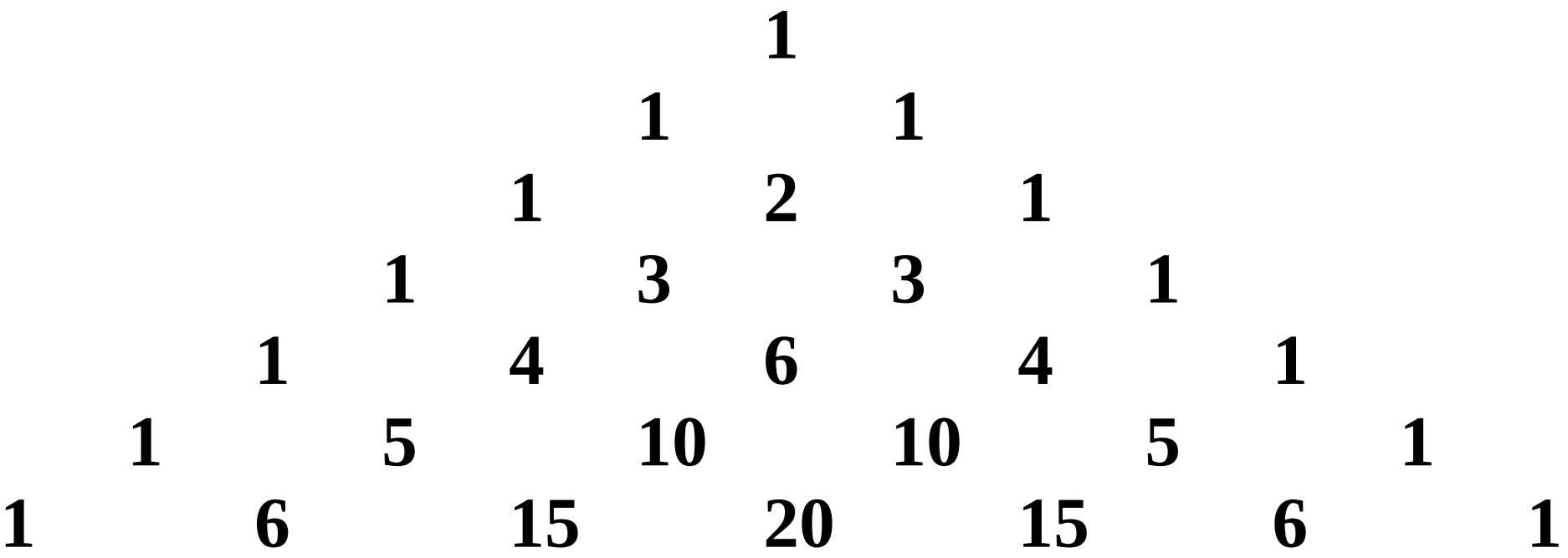
$C[n,k] =_{df} n!/(k!*(n-k)!)$, dla $0 <= k <= n$

$C[n,k] = 1$, dla $k = 0$ i $k = n$

$C[n,k] = n(n-1)...(n-k+1)/k(k-1)...k$, dla $k > 0$

$C[n,k] = C[n-1,k-1] + C[n-1,k]$, dla $0<k<n$

Trójkąt Pascala



Zadanie: dla danego n >= 0 obliczyć trójkąt Pascala do poziomu n

Podejście 1 – tablica dwuwymiarowa

```
//inicjalizacja
for (i = 0; i <= n; i++){
        Pascal[i][0] = 1; Pascal[i][i] = 1;
    }

 //obliczanie wnętrza trójkąta
 for (i = 2; i <= n; i++)
      for (j = 1; j <= i-1; j++)
            Pascal[i][j] = Pascal[i-1][j-1] + Pascal[i-1][j];
```

Podejście 2 – tablica jednowymiarowa

```
//liczenie trójkąta Pascala wierszami
   Pascal[0] = 1;
   for (i = 0; i <= n; i++){
      Pascal[i] = 1;
      for (j = i-1; j > 0; j--)
         Pascal[j] = Pascal[j-1] + Pascal[j];
      //wypisanie kolejnego wiersza
      for (j = 0; j <= i; j++)
          cout << setw(5) << Pascal [j];
      cout << endl;
   }
```

Zastosowanie podejścia 2 do obliczania współczynnika C[n,k]

```
//liczenie trójkąta Pascala wierszami
//aż do wiersza n
Pascal[0] = 1;
for (i = 0; i <= n; i++){
    Pascal[i] = 1;
    for (j = i-1; j > 0; j--)
       Pascal[j] = Pascal[j-1] + Pascal[j];
}

// C[n,k] = Pascal[k]
```

Uwaga na duże liczby:

liczba bitów	typ	zakres
8	char	-128 : +127
	bez znaku	0 : 255
16	short int	−32 768: +32 767
	bez znaku	0 : 65 535
32	int, long	−2 147 483 648 : +2 147 483 647
	bez znaku	0 : +4 294 967 295
64	long long	−9 223 372 036 854 775 808 : +9 223 372 036 854 775 807
	bez znaku	0 : +18 446 744 073 709 551 615

Oto obliczanie współczynnika dwumianowego z prostą arytmetyką dużych liczb:

```
//*****************************************************
// prosta arytmetyka dużych liczb
const int zakres = 101;

void zero(int a[]){
//a = 0;
   int i; for (i = 0; i < zakres; i++) a[i] = 0;
}
void jeden(int a[]){
//a = 1;
...
}
void dodaj(int a[], int b[], int c[]){
//c = a + b;
   int i, p, r;
   p = 0;
   for (i = 0; i < zakres; i++){
      r = (a[i] + b[i] + p)%10;
      p = (a[i] + b[i] + p)/10;
      c[i] = r;
   }
}
```

```
void wypisz(int a[]){
//wypisz a;
    int i,j;
    for (i = zakres -1; (a[i] == 0) && (i > 0); i--);
    for (j = i; j >= 0; j--)
        cout << a[j];
    cout << endl;
}

//**************************************************
```

Sam algorytm:

```
//wczytaj dane
    cin >> n >> k;

    //liczenie trójkąta Pascala wierszami
    //aż do wiersza n
    jeden(Pascal[0]);
    for (i = 0; i <= n; i++){
        jeden(Pascal[i]);
        for (j = i-1; j > 0; j--)
            dodaj(Pascal[j-1], Pascal[j], Pascal[j]);
    }

    // C[n,k] = Pascal[k]
    wypisz(Pascal[k]);
```

Liczby Stirlinga II rodzaju

S[n,k] – liczba podziałów zbioru n-elementowego na dokładnie k podzbiorów

Zadanie: Dla danych liczb 0<=k<=n wyznacz liczbę Stirlinga S[n,k].

Liczby Bella

B[n] – liczba wszystkich podziałów zbioru n-elementowego

$B[n] = \Sigma_{n,k} S[n,k]$

Liczbę B[n] można policzyć wyznaczając najpierw n-ty wiersz „trójkąta Bella”

Inny sposób liczenia liczb Bella.

Korzystamy ze wzoru: $B[n] = \Sigma_{0<=k<=n-1} C[n-1,k]B[k]$, dla $n > 0$.

Algorytm magiczny

```
B[0][0] = 1;
for (i = 1; i <= n-1; i++){
    B[0][i] = B[i-1][i-1];
    for (j = 1; j <= i; j++)
       B[j][i] = B[j-1][i-1] + B[j-1][i];
}
```

B[n-1,n-1] jest szukaną liczbą.

Algorytm czarnoksiężnika

```
B[0] = 1;
for (i = 1; i <= n-1; i++){
    c = B[i-1];
    for (j = 1; j <= i; j++){
      d = B[j-1] + c;
      B[j-1] = c;
      c = d;
    }
    B[i] = c;
}
```