Sparse Matrix

→ Very few non-zero element in Matrix

row	column	no of non-zero element ← coordinate list
Total rows	Total column	Total num of non-zero element

Compressed sparse row (30% of memory is reduced)

A [x, x, - - - - -] → element

IA [0, 1, 5, - - - -] → Row of element

JA [5, 8, 2 - - - -] → column of element

→ Addition of sparse matrix

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 5 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

↓

	0	1	2	
A	3	1	2	→ row
Total rows →	3	1	2	→ column
Total column →	2	1	6	→ element

	0	1	2
B	3	1	2
	3	2	2
	2	5	4

Total rows + new rows + element

	0	1	2	3
C	3	1	1	2
	3	1	2	2
	3	1	5	10

```
* Struct Element
{
  int i;
  int j;
  int e;
}

* Struct sparse
{
  int m;
  int n;
  int num;
  Struct element *e;

int main()
{
  Struct sparse s;
  create(&s);

create(struct sparse *s)
{
  #take dimension in s->m & s->n
  #take number of input in s->num
  s->e = new Element [s->num]
  for(i=0; i<s->num; i++)
  { scanf("%d %d %d", &s->e[i].i, &s->e[i].j, &s->e[i].z);
```

* Adding.

Make two sparse matrix s1, s2

in function add

```
add (struct sparse*s1, struct sparse *s2)
{
  if (s1->m != s2->m || s2->n != s2->n) // dimension is different
      return 0;
  sum = new Sparse // struct sparse in heap
  sum->m = s1->m; // giving structure
  sum->n = s1->n // dimensions
  sum->e = new Element[s1->num + s2->num]
```

// Adding algorithm

i=0, j=0, K=0;
↳ for s1 ↳ for s2 ↳ for sum

while (i < s1→num && j < s2→num)
{
if (s1→e[i].i < s2→e[j].i) // if row of s1 is less than s2 copy s1 to sum
sum→e[k++] = s1→e[i++]; // copy complete structure.
else if (s1→e[i].i > s2.e[j].i)
sum→e[k++] = s2.e[j++];
else
if (s1→e[i].j < s2→e[j].j)
sum→e[k++] = s1→e[i++];
elif (s1→e[i].j > s2→e[j].j) // if column of s2 is less than s1 copy s2 to sum
sum→e[k++] = s1→e[j++];
else
{sum→e[k++] = s1→e[i++];
sum→e[k++].x = s2.e[j++].x;

// if row of s2 is less than s1 copy struct s2 to sum

// if column of s1 is less than s2 copy struct s1 to sum

if both row & column are equal first copy entires structure of s1 & then add value of structure of s2 to sum structure value

friend function is global (no need of scope resolution operator

sparse operator + (sparse &s);

istream → for input

ostream → for output

istream should return reference var is

reference ↓ function name ↓

format → istream & operator >> (istream &is, sparse &s)

ostream & operator << (ostream &os, sparse &s)

↳ cin >> class name

cout << class name

Sparse Sparse::operator + (sparse &s)

↑ return type

XXX

Polynomial Representation

$p(x) = 3x^5 + 2x^4 + 5x^2 + 2x + 7$

(5 → EXP, 3 → coeff)

coeff	3	2	5	2	7
Exp	5	4	2	1	0

```
struct poly
{
   int n;
   Struct Term *t;
};
```

```
Struct Term {
int coeff;
int Exp;
};
```

Algo →

i) input >> number of elements.
(Cin, polydata type variable)
in variable n of poly)

ii) Assign heap memory to t as array
Eg. p.t = new Term[p.n]

evaluation of $3x^2 + 5x$

$= 3 \times x \times x + 5 \times x$

iii) for i=0 till i<p.n

input (p.t[i].coeff)

input (p.t[i].Exp)

- Evaluation of $3x^2 + 5x$ = $3 \times x \times x + 5 \times x$
- Addition of polynomial (see in repo for code)