**Увеличение быстродействия построения ilu(p) сглаживателя алгебраического многосеточного метода**.

Иванов Кирилл Андреевич

МАИ, Россия, Москва

email: kirill7785@mail.ru

**Введение**

Задачи вычислительной теплопередачи приводят к системам линейных алгебраических уравнений (СЛАУ) большого размера, например, 5\*107 неизвестных. Для решения подобных СЛАУ автор предлагает использовать алгоритм BiCGStab, предобусловленный классическим алгебраическим многосеточным методом (CAMG). В качестве сглаживателя в CAMG предлагается использовать ilu(p) разложение. Ilu(p) разложение реализовано в библиотеке SPARSKIT2 Ю.Саада. Узким местом предлагаемого алгоритма ilu(p) разложения является строка с линейным поиском минимального элемента в списке. Список - структура данных в информатике. Замедление быстродействия построения ilu(p) разложения проявляется на матрицах с большим значением nnz/n, например 60-125. Здесь nnz – число ненулевых коэффициентов в матрице, n – число неизвестных. Данные матрицы с большим значением nnz/n создаются CAMG алгоритмом на средних по «глубине» уровнях, построенной иерархии сетки.

Линейный поиск предлагается заменить на комбинацию двоичной кучи и хеш таблицы. Двоичная куча и хеш таблица – структуры данных в информатике. Двоичная куча позволяет за время O(log2(n)) извлекать наименьший элемент. Для поддержания порядка кучи используются вспомогательные операции fixUp и fixDouwn с быстродействием O(log2(n)) каждая. Здесь n – число элементов в куче. Для быстрого доступа к элементу кучи по значению ключа используется соединение кучи и хеш таблицы (См. в тексте статьи). Данная статья посвящена описанию предлагаемого подхода.

**Алгоритм ilu(p) разложения. Строка кода с низким быстродействием – линейным поиском**.

*Листинг 1.* Алгоритм из библиотеки SPARSKIT2 Ю.Саада.

// Сдержит медленный линейный поиск и из-за этого непригодна.

/\* ----------------------------------------------------------------------- \*/

/\* Subroutine \*/ integer iluk\_(integer n, doublereal\* &a, integer\* &ja, integer\* &ia,

integer lfil, doublereal\* &alu, integer\* &jlu, integer\* &ju,

integer\* &levs, integer iwk, doublereal\* &w, integer\* &jw, integer &ierr)

{

/\* System generated locals \*/

integer i\_\_1, i\_\_2, i\_\_3, i\_\_4;

/\* Local variables \*/

integer i\_\_, j, k;

doublereal s, t;

integer j1, j2, n2, ii, jj, ju0;

doublereal fact;

integer lenl, jlev, lenu, jpos, jrow;

/\* ----------------------------------------------------------------------\* \*/

/\* SPARSKIT ROUTINE ILUK -- ILU WITH LEVEL OF FILL-IN OF K (ILU(k)) \* \*/

/\* ----------------------------------------------------------------------\* \*/

/\* on entry: \*/

/\* ========== \*/

/\* n = integer. The row dimension of the matrix A. The matrix \*/

/\* a,ja,ia = matrix stored in Compressed Sparse Row format. \*/

/\* lfil = integer. The fill-in parameter. Each element whose \*/

/\* leve-of-fill exceeds lfil during the ILU process is dropped. \*/

/\* lfil must be .ge. 0 \*/

/\* tol = real\*8. Sets the threshold for dropping small terms in the \*/

/\* factorization. See below for details on dropping strategy. \*/

/\* iwk = integer. The minimum length of arrays alu, jlu, and levs. \*/

/\* On return: \*/

/\* =========== \*/

/\* alu,jlu = matrix stored in Modified Sparse Row (MSR) format containing \*/

/\* the L and U factors together. The diagonal (stored in \*/

/\* alu(1:n) ) is inverted. Each i-th row of the alu,jlu matrix \*/

/\* contains the i-th row of L (excluding the diagonal entry=1) \*/

/\* followed by the i-th row of U. \*/

/\* ju = integer array of length n containing the pointers to \*/

/\* the beginning of each row of U in the matrix alu,jlu. \*/

/\* levs = integer (work) array of size iwk -- which contains the \*/

/\* levels of each element in alu, jlu. \*/

/\* ierr = integer. Error message with the following meaning. \*/

/\* ierr = 0 --> successful return. \*/

/\* ierr .gt. 0 --> zero pivot encountered at step number ierr. \*/

/\* ierr = -1 --> Error. input matrix may be wrong. \*/

/\* (The elimination process has generated a \*/

/\* row in L or U whose length is .gt. n.) \*/

/\* ierr = -2 --> The matrix L overflows the array al. \*/

/\* ierr = -3 --> The matrix U overflows the array alu. \*/

/\* ierr = -4 --> Illegal value for lfil. \*/

/\* ierr = -5 --> zero row encountered in A or U. \*/

/\* work arrays: \*/

/\* ============= \*/

/\* jw = integer work array of length 3\*n. \*/

/\* w = real work array of length n \*/

/\* Notes/known bugs: This is not implemented efficiently storage-wise. \*/

/\* For example: Only the part of the array levs(\*) associated with \*/

/\* the U-matrix is needed in the routine.. So some storage can \*/

/\* be saved if needed. The levels of fills in the LU matrix are \*/

/\* output for information only -- they are not needed by LU-solve. \*/

/\* ---------------------------------------------------------------------- \*/

/\* w, ju (1:n) store the working array [1:ii-1 = L-part, ii:n = u] \*/

/\* jw(n+1:2n) stores the nonzero indicator. \*/

/\* Notes: \*/

/\* ------ \*/

/\* All the diagonal elements of the input matrix must be nonzero. \*/

/\* ----------------------------------------------------------------------\* \*/

/\* locals \*/

/\* Parameter adjustments \*/

--jw;

--w;

--ju;

--ia;

--a;

--ja;

--alu;

--jlu;

--levs;

/\* Function Body \*/

if (lfil < 0) {

goto L998;

}

/\* ----------------------------------------------------------------------- \*/

/\* initialize ju0 (points to next element to be added to alu,jlu) \*/

/\* and pointer array. \*/

/\* ----------------------------------------------------------------------- \*/

n2 = n + n;

ju0 = n + 2;

jlu[1] = ju0;

/\* initialize nonzero indicator array + levs array -- \*/

i\_\_1 = n << 1;

for (j = 1; j <= i\_\_1; ++j) {

jw[j] = 0;

/\* L1: \*/

}

/\* ----------------------------------------------------------------------- \*/

/\* beginning of main loop. \*/

/\* ----------------------------------------------------------------------- \*/

i\_\_1 = n;

for (ii = 1; ii <= i\_\_1; ++ii) {

j1 = ia[ii];

j2 = ia[ii + 1] - 1;

/\* unpack L-part and U-part of row of A in arrays w \*/

lenu = 1;

lenl = 0;

jw[ii] = ii;

w[ii] = 0.f;

jw[n + ii] = ii;

i\_\_2 = j2;

for (j = j1; j <= i\_\_2; ++j) {

k = ja[j];

t = a[j];

if (t == 0.f) {

goto L170;

}

if (k < ii) {

++lenl;

jw[lenl] = k;

w[lenl] = t;

jw[n2 + lenl] = 0;

jw[n + k] = lenl;

} else if (k == ii) {

w[ii] = t;

jw[n2 + ii] = 0;

} else {

++lenu;

jpos = ii + lenu - 1;

jw[jpos] = k;

w[jpos] = t;

jw[n2 + jpos] = 0;

jw[n + k] = jpos;

}

L170:

;

}

jj = 0;

/\* eliminate previous rows \*/

L150:

++jj;

if (jj > lenl) {

goto L160;

}

/\* ----------------------------------------------------------------------- \*/

/\* in order to do the elimination in the correct order we must select \*/

/\* the smallest column index among jw(k), k=jj+1, ..., lenl. \*/

/\* ----------------------------------------------------------------------- \*/

jrow = jw[jj];

k = jj;

/\* determine smallest column index \*/

i\_\_2 = lenl;

// Дьявольски медленный поиск минимума. Это линейный поиск.

//printf("jj=%d\n",jj);// jj==1 далеко не всегда.

// Это означает что нужно поддерживать удаление элемента по ключу.

//getchar();

for (j = jj + 1; j <= i\_\_2; ++j) {

if (jw[j] < jrow) {

jrow = jw[j];

k = j;

}

/\* L151: \*/

}

if (k != jj) {

/\* exchange in jw \*/

j = jw[jj];

jw[jj] = jw[k];

jw[k] = j;

/\* exchange in jw(n+ (pointers/ nonzero indicator). \*/

jw[n + jrow] = jj;

jw[n + j] = k;

/\* exchange in jw(n2+ (levels) \*/

j = jw[n2 + jj];

jw[n2 + jj] = jw[n2 + k];

jw[n2 + k] = j;

/\* exchange in w \*/

s = w[jj];

w[jj] = w[k];

w[k] = s;

}

/\* zero out element in row by resetting jw(n+jrow) to zero. \*/

jw[n + jrow] = 0;

/\* get the multiplier for row to be eliminated (jrow) + its level \*/

fact = w[jj] \* alu[jrow];

jlev = jw[n2 + jj];

if (jlev > lfil) {

goto L150;

}

/\* combine current row and row jrow \*/

i\_\_2 = jlu[jrow + 1] - 1;

for (k = ju[jrow]; k <= i\_\_2; ++k) {

s = fact \* alu[k];

j = jlu[k];

jpos = jw[n + j];

if (j >= ii) {

/\* dealing with upper part. \*/

if (jpos == 0) {

/\* this is a fill-in element \*/

++lenu;

if (lenu > n) {

goto L995;

}

i\_\_ = ii + lenu - 1;

jw[i\_\_] = j;

jw[n + j] = i\_\_;

w[i\_\_] = -s;

jw[n2 + i\_\_] = jlev + levs[k] + 1;

} else {

/\* this is not a fill-in element \*/

w[jpos] -= s;

/\* Computing MIN \*/

i\_\_3 = jw[n2 + jpos], i\_\_4 = jlev + levs[k] + 1;

jw[n2 + jpos] = min(i\_\_3,i\_\_4);

}

} else {

/\* dealing with lower part. \*/

if (jpos == 0) {

/\* this is a fill-in element \*/

++lenl;

if (lenl > n) {

goto L995;

}

jw[lenl] = j;

jw[n + j] = lenl;

w[lenl] = -s;

jw[n2 + lenl] = jlev + levs[k] + 1;

} else {

/\* this is not a fill-in element \*/

w[jpos] -= s;

/\* Computing MIN \*/

i\_\_3 = jw[n2 + jpos], i\_\_4 = jlev + levs[k] + 1;

jw[n2 + jpos] = min(i\_\_3,i\_\_4);

}

}

/\* L203: \*/

}

w[jj] = fact;

jw[jj] = jrow;

goto L150;

L160:

/\* reset double-pointer to zero (U-part) \*/

i\_\_2 = lenu;

for (k = 1; k <= i\_\_2; ++k) {

jw[n + jw[ii + k - 1]] = 0;

/\* L308: \*/

}

/\* update l-matrix \*/

i\_\_2 = lenl;

for (k = 1; k <= i\_\_2; ++k) {

if (ju0 > iwk) {

goto L996;

}

if (jw[n2 + k] <= lfil) {

alu[ju0] = w[k];

jlu[ju0] = jw[k];

++ju0;

}

/\* L204: \*/

}

/\* save pointer to beginning of row ii of U \*/

ju[ii] = ju0;

/\* update u-matrix \*/

i\_\_2 = ii + lenu - 1;

for (k = ii + 1; k <= i\_\_2; ++k) {

if (jw[n2 + k] <= lfil) {

jlu[ju0] = jw[k];

alu[ju0] = w[k];

levs[ju0] = jw[n2 + k];

++ju0;

}

/\* L302: \*/

}

if (fabs(w[ii]) < 1.0e-30) {

printf("w[%d]=%e\n",ii,w[ii]);

goto L999;

}

alu[ii] = 1.0 / w[ii];

/\* update pointer to beginning of next row of U. \*/

jlu[ii + 1] = ju0;

/\* ----------------------------------------------------------------------- \*/

/\* end main loop \*/

/\* ----------------------------------------------------------------------- \*/

/\* L500: \*/

}

++jw;

++w;

++ju;

++ia;

++a;

++ja;

++alu;

++jlu;

++levs;

ierr = 0;

return 0;

/\* incomprehensible error. Matrix must be wrong. \*/

L995:

++jw;

++w;

++ju;

++ia;

++a;

++ja;

++alu;

++jlu;

++levs;

ierr = -1;

return 0;

/\* insufficient storage in L. \*/

L996:

++jw;

++w;

++ju;

++ia;

++a;

++ja;

++alu;

++jlu;

++levs;

ierr = -2;

return 0;

/\* insufficient storage in U. \*/

/\* L997: \*/

// ierr = -3;

// return 0;

/\* illegal lfil entered. \*/

L998:

++jw;

++w;

++ju;

++ia;

++a;

++ja;

++alu;

++jlu;

++levs;

ierr = -4;

return 0;

/\* zero row encountered in A or U. \*/

L999:

++jw;

++w;

++ju;

++ia;

++a;

++ja;

++alu;

++jlu;

++levs;

ierr = -5;

return 0;

/\* ----------------end-of-iluk-------------------------------------------- \*/

/\* ----------------------------------------------------------------------- \*/

} /\* iluk\_ \*/

**Способ замены линейного поиска на комбинацию двоичной кучи и хеш таблицы**.

Используется следующий интерфейс абстрактного типа данных:

*Листинг 2.*

// Очередь по приоритетам в соединении с хеш таблицей.

//PQ(integer maxN, integer max\_key\_size); // конструктор

//~PQ(); // деструктор

// Есть ли элемент с данным ключём в таблице ?

//bool isfound(integer key);

//bool empty() const; // проверка на пустоту.

// Очищаем содержимое и она снова готова к использованию.

//void clear();

// Вернуть элемент с заданным ключём:

// Обязательно предполагается что ключ существует внутри таблицы.

//Item get(integer key);

// прочитать максимальный элемент.

//Item readmax();

// Прочитать значение ключа максимального элемента.

//integer readkeymaxelm();

// Вставить элемент item в очередь по

// приоритетам если элемент item имеет ключ key.

//template <class Item>

//void insert(Item item, integer key);

// Возвратить максимальный элемент

// и удалить его.

//Item getmax();

// Заменяет элемент с ключём key на элемент val с тем же ключём key.

// При этом ключ key должен быть уникальным.

//void modify(integer key, Item val);

// Удаление элемента с заданным значением ключа.

//void remove(integer key);

// У элемента изменть значение старого ключа на новый ключ

// при этом меняется и само содержимое элемента.

//void change(integer key\_serch, integer key\_new, integer item\_new);

// Меняет местами значения элементов с ключами i и j.

// Сохраняет порядок кучи.

//void exchange(integer i, integer j);

Реализация ilu(p) разложения, использующая описанный абстрактный тип данных.

*Листинг 3.*

// Медленный линейный поиск ликвидирован.

/\* ----------------------------------------------------------------------- \*/

/\* Subroutine \*/ integer iluk\_(integer n, doublereal\* &a, integer\* &ja, integer\* &ia,

integer lfil, doublereal\* &alu, integer\* &jlu, integer\* &ju,

integer\* &levs, integer iwk, doublereal\* &w, integer\* &jw, integer &ierr)

{

/\* System generated locals \*/

integer i\_\_1, i\_\_2, i\_\_3, i\_\_4;

/\* Local variables \*/

integer i\_\_, j, k;

doublereal s, t;

integer j1, j2, n2, ii, jj, ju0;

doublereal fact;

integer lenl, jlev, lenu, jpos, jrow;

// Переменные для проверки корректности исходного кода.

integer jrow1, jrow2, k1, k2;

/\* ----------------------------------------------------------------------\* \*/

/\* SPARSKIT ROUTINE ILUK -- ILU WITH LEVEL OF FILL-IN OF K (ILU(k)) \* \*/

/\* ----------------------------------------------------------------------\* \*/

/\* on entry: \*/

/\* ========== \*/

/\* n = integer. The row dimension of the matrix A. The matrix \*/

/\* a,ja,ia = matrix stored in Compressed Sparse Row format. \*/

/\* lfil = integer. The fill-in parameter. Each element whose \*/

/\* leve-of-fill exceeds lfil during the ILU process is dropped. \*/

/\* lfil must be .ge. 0 \*/

/\* tol = real\*8. Sets the threshold for dropping small terms in the \*/

/\* factorization. See below for details on dropping strategy. \*/

/\* iwk = integer. The minimum length of arrays alu, jlu, and levs. \*/

/\* On return: \*/

/\* =========== \*/

/\* alu,jlu = matrix stored in Modified Sparse Row (MSR) format containing \*/

/\* the L and U factors together. The diagonal (stored in \*/

/\* alu(1:n) ) is inverted. Each i-th row of the alu,jlu matrix \*/

/\* contains the i-th row of L (excluding the diagonal entry=1) \*/

/\* followed by the i-th row of U. \*/

/\* ju = integer array of length n containing the pointers to \*/

/\* the beginning of each row of U in the matrix alu,jlu. \*/

/\* levs = integer (work) array of size iwk -- which contains the \*/

/\* levels of each element in alu, jlu. \*/

/\* ierr = integer. Error message with the following meaning. \*/

/\* ierr = 0 --> successful return. \*/

/\* ierr .gt. 0 --> zero pivot encountered at step number ierr. \*/

/\* ierr = -1 --> Error. input matrix may be wrong. \*/

/\* (The elimination process has generated a \*/

/\* row in L or U whose length is .gt. n.) \*/

/\* ierr = -2 --> The matrix L overflows the array al. \*/

/\* ierr = -3 --> The matrix U overflows the array alu. \*/

/\* ierr = -4 --> Illegal value for lfil. \*/

/\* ierr = -5 --> zero row encountered in A or U. \*/

/\* work arrays: \*/

/\* ============= \*/

/\* jw = integer work array of length 3\*n. \*/

/\* w = real work array of length n \*/

/\* Notes/known bugs: This is not implemented efficiently storage-wise. \*/

/\* For example: Only the part of the array levs(\*) associated with \*/

/\* the U-matrix is needed in the routine.. So some storage can \*/

/\* be saved if needed. The levels of fills in the LU matrix are \*/

/\* output for information only -- they are not needed by LU-solve. \*/

/\* ---------------------------------------------------------------------- \*/

/\* w, ju (1:n) store the working array [1:ii-1 = L-part, ii:n = u] \*/

/\* jw(n+1:2n) stores the nonzero indicator. \*/

/\* Notes: \*/

/\* ------ \*/

/\* All the diagonal elements of the input matrix must be nonzero. \*/

/\* ----------------------------------------------------------------------\* \*/

/\* locals \*/

/\* Parameter adjustments \*/

--jw;

--w;

--ju;

--ia;

--a;

--ja;

--alu;

--jlu;

--levs;

// Целочисленная очередь по приоритетам.

PQ<integer> pq(n + 1, n + 1);

/\* Function Body \*/

if (lfil < 0) {

goto L998;

}

/\* ----------------------------------------------------------------------- \*/

/\* initialize ju0 (points to next element to be added to alu,jlu) \*/

/\* and pointer array. \*/

/\* ----------------------------------------------------------------------- \*/

n2 = n + n;

ju0 = n + 2;

jlu[1] = ju0;

/\* initialize nonzero indicator array + levs array -- \*/

i\_\_1 = n << 1;

for (j = 1; j <= i\_\_1; ++j) {

jw[j] = 0;

/\* L1: \*/

}

/\* ----------------------------------------------------------------------- \*/

/\* beginning of main loop. \*/

/\* ----------------------------------------------------------------------- \*/

i\_\_1 = n;

for (ii = 1; ii <= i\_\_1; ++ii) {

j1 = ia[ii];

j2 = ia[ii + 1] - 1;

/\* unpack L-part and U-part of row of A in arrays w \*/

lenu = 1;

lenl = 0;

jw[ii] = ii;

w[ii] = 0.f;

jw[n + ii] = ii;

i\_\_2 = j2;

for (j = j1; j <= i\_\_2; ++j) {

k = ja[j];

t = a[j];

if (t == 0.f) {

goto L170;

}

if (k < ii) {

++lenl;

jw[lenl] = k;

// push jw

pq.insert(-k, lenl);

w[lenl] = t;

jw[n2 + lenl] = 0;

jw[n + k] = lenl;

}

else if (k == ii) {

w[ii] = t;

jw[n2 + ii] = 0;

}

else {

++lenu;

jpos = ii + lenu - 1;

jw[jpos] = k;

w[jpos] = t;

jw[n2 + jpos] = 0;

jw[n + k] = jpos;

}

L170:

;

}

jj = 0;

/\* eliminate previous rows \*/

L150:

// remove in jw

pq.remove(jj);

++jj;

if (jj > lenl) {

goto L160;

}

/\* ----------------------------------------------------------------------- \*/

/\* in order to do the elimination in the correct order we must select \*/

/\* the smallest column index among jw(k), k=jj+1, ..., lenl. \*/

/\* ----------------------------------------------------------------------- \*/

jrow = jw[jj];

k = jj;

/\* determine smallest column index \*/

i\_\_2 = lenl;

if (0) {

// Дьявольски медленный поиск минимума. Это линейный поиск.

//printf("jj=%d\n",jj);// jj==1 далеко не всегда.

// Это означает, что нужно поддерживать удаление элемента по ключу.

// Чрезвычайно медленный линейный поиск.

for (j = jj + 1; j <= i\_\_2; ++j) {

if (jw[j] < jrow) {

jrow = jw[j];

k = j;

}

}

}

else {

if (0) {

// Проверочный участок кода.

// Раскоментировать если нужно проверить код.

jrow1 = jrow;

k1 = k;

for (j = jj + 1; j <= i\_\_2; ++j) {

if (jw[j] < jrow1) {

jrow1 = jw[j];

k1 = j;

}

}

if (jj + 1 <= i\_\_2) {

// remove in jw

pq.remove(jj);

integer jrow3 = jrow, k3 = k;

// на основе очереди по приоритетам в сочетании с хеш таблицей.

if (jrow > (-pq.readmax())) {

jrow = -pq.readmax();

k = pq.readkeymaxelm();

}

pq.insert(-jrow3, k3);

}

jrow2 = jrow;

k2 = k;

if ((jrow1 != jrow2) || (k1 != k2)) {

for (j = jj; j <= i\_\_2; ++j) {

printf("jw[%d]=%d ", j, jw[j]);

}

printf("\n");

pq.print\_log('s');

printf("k1=%d k2=%d jrow1=%d jrow2=%d\n", k1, k2, jrow1, jrow2);

getchar();

}

}

else {

jrow = -pq.readmax();

k = pq.readkeymaxelm();

/\*

if (jj + 1 <= i\_\_2) {

// remove in jw

pq.remove(jj);

integer jrow3 = jrow, k3 = k;

// на основе очереди по приоритетам в сочетании с хеш таблицей.

if (jrow > (-pq.readmax())) {

jrow = -pq.readmax();

k = pq.readkeymaxelm();

}

pq.insert(-jrow3, k3);

}

\*/

}

}

if (k != jj) {

/\* exchange in jw \*/

j = jw[jj];

jw[jj] = jw[k];

jw[k] = j;

/\* exchange in jw(n+ (pointers/ nonzero indicator). \*/

jw[n + jrow] = jj;

jw[n + j] = k;

/\* exchange in jw(n2+ (levels) \*/

j = jw[n2 + jj];

jw[n2 + jj] = jw[n2 + k];

jw[n2 + k] = j;

/\* exchange in w \*/

s = w[jj];

w[jj] = w[k];

w[k] = s;

pq.exchange\_speshial\_for\_Saad(jj, k);

}

/\* zero out element in row by resetting jw(n+jrow) to zero. \*/

jw[n + jrow] = 0;

/\* get the multiplier for row to be eliminated (jrow) + its level \*/

fact = w[jj] \* alu[jrow];

jlev = jw[n2 + jj];

if (jlev > lfil) {

goto L150;

}

/\* combine current row and row jrow \*/

i\_\_2 = jlu[jrow + 1] - 1;

for (k = ju[jrow]; k <= i\_\_2; ++k) {

s = fact \* alu[k];

j = jlu[k];

jpos = jw[n + j];

if (j >= ii) {

/\* dealing with upper part. \*/

if (jpos == 0) {

/\* this is a fill-in element \*/

++lenu;

if (lenu > n) {

goto L995;

}

i\_\_ = ii + lenu - 1;

jw[i\_\_] = j;

jw[n + j] = i\_\_;

w[i\_\_] = -s;

jw[n2 + i\_\_] = jlev + levs[k] + 1;

}

else {

/\* this is not a fill-in element \*/

w[jpos] -= s;

/\* Computing MIN \*/

i\_\_3 = jw[n2 + jpos], i\_\_4 = jlev + levs[k] + 1;

jw[n2 + jpos] = min(i\_\_3, i\_\_4);

}

}

else {

/\* dealing with lower part. \*/

if (jpos == 0) {

/\* this is a fill-in element \*/

++lenl;

if (lenl > n) {

goto L995;

}

// push jw

pq.insert(-j, lenl);

jw[lenl] = j;

jw[n + j] = lenl;

w[lenl] = -s;

jw[n2 + lenl] = jlev + levs[k] + 1;

}

else {

/\* this is not a fill-in element \*/

w[jpos] -= s;

/\* Computing MIN \*/

i\_\_3 = jw[n2 + jpos], i\_\_4 = jlev + levs[k] + 1;

jw[n2 + jpos] = min(i\_\_3, i\_\_4);

}

}

/\* L203: \*/

}

w[jj] = fact;

//pq.remove(jj);

//pq.insert(-jrow, jj);

jw[jj] = jrow;

goto L150;

L160:

/\* reset double-pointer to zero (U-part) \*/

i\_\_2 = lenu;

for (k = 1; k <= i\_\_2; ++k) {

jw[n + jw[ii + k - 1]] = 0;

/\* L308: \*/

}

/\* update l-matrix \*/

i\_\_2 = lenl;

for (k = 1; k <= i\_\_2; ++k) {

if (ju0 > iwk) {

goto L996;

}

if (jw[n2 + k] <= lfil) {

alu[ju0] = w[k];

jlu[ju0] = jw[k];

++ju0;

}

/\* L204: \*/

}

/\* save pointer to beginning of row ii of U \*/

ju[ii] = ju0;

/\* update u-matrix \*/

i\_\_2 = ii + lenu - 1;

for (k = ii + 1; k <= i\_\_2; ++k) {

if (jw[n2 + k] <= lfil) {

jlu[ju0] = jw[k];

alu[ju0] = w[k];

levs[ju0] = jw[n2 + k];

++ju0;

}

/\* L302: \*/

}

if (fabs(w[ii]) < 1.0e-30) {

printf("w[%d]=%e\n", ii, w[ii]);

//w[ii] = 1.0;

//printf("k1=%d k2=%d jrow1=%d jrow2=%d\n",k1,k2,jrow1,jrow2);

getchar();

goto L999;

}

alu[ii] = 1.0 / w[ii];

/\* update pointer to beginning of next row of U. \*/

jlu[ii + 1] = ju0;

/\* ----------------------------------------------------------------------- \*/

/\* end main loop \*/

/\* ----------------------------------------------------------------------- \*/

/\* L500: \*/

}

++jw;

++w;

++ju;

++ia;

++a;

++ja;

++alu;

++jlu;

++levs;

ierr = 0;

return 0;

/\* incomprehensible error. Matrix must be wrong. \*/

L995:

++jw;

++w;

++ju;

++ia;

++a;

++ja;

++alu;

++jlu;

++levs;

ierr = -1;

return 0;

/\* insufficient storage in L. \*/

L996:

++jw;

++w;

++ju;

++ia;

++a;

++ja;

++alu;

++jlu;

++levs;

ierr = -2;

return 0;

/\* insufficient storage in U. \*/

/\* L997: \*/

// ierr = -3;

// return 0;

/\* illegal lfil entered. \*/

L998:

++jw;

++w;

++ju;

++ia;

++a;

++ja;

++alu;

++jlu;

++levs;

ierr = -4;

return 0;

/\* zero row encountered in A or U. \*/

L999:

++jw;

++w;

++ju;

++ia;

++a;

++ja;

++alu;

++jlu;

++levs;

ierr = -5;

return 0;

/\* ----------------end-of-iluk-------------------------------------------- \*/

/\* ----------------------------------------------------------------------- \*/

} /\* iluk\_quick \*/

**Компьютерная реализация двоичной кучи с хеш таблицей**.

*Листинг 4.*

// Используется в алгебраическом многосеточном методе.

// Используются только следующие функции:

// clear, remove, insert, readkeymaxelm.

// Соединяем с быстродействующей хеш таблицей.

template <class Item>

void exch(integer i, integer j, Item\* &pq, integer\* &qp, integer\* &hash) {

// exchange

/\*

// begin

integer t;

Item buf1 = pq[qp[i]];

Item buf2 = pq[qp[j]];

t = qp[i];

qp[i] = qp[j];

qp[j] = t;

pq[qp[i]] = buf1;

pq[qp[j]] = buf2;

// end

\*/

//printf("exchange\n");

Item t;

t = pq[j];

pq[j] = pq[i];

pq[i] = t;

integer p;

p = hash[qp[i]];

hash[qp[i]] = hash[qp[j]];

hash[qp[j]] = p;

p = qp[j];

qp[j] = qp[i];

qp[i] = p;

}

// Восходящая установка структуры сортирующего дерева.

// Роберт Седжвик с. 366 в книге 2002 года.

template <class Item>

void fixUp(Item\* &a, integer\* &inda, integer\* &hash, integer k)

{

while (k > 1 && a[k / 2] < a[k])

{

integer kdiv2 = k / 2;

exch(k, kdiv2, a, inda, hash);

k = kdiv2;

}

}

// Нисходящая установка структуры сортирующего дерева.

template <class Item>

void fixDown(Item\* &a, integer\* &inda, integer\* &hash, integer k, integer N)

{

while (2 \* k <= N)

{

integer j = 2 \* k;

if (j < N&&a[j] < a[j + 1]) j++;

if (!(a[k] < a[j])) break;

exch(k, j, a, inda, hash);

k = j;

}

}

//PQ(integer maxN, integer max\_key\_size);

//~PQ();

// Есть ли элемент с данным ключём в таблице ?

//bool isfound(integer key);

//bool empty() const;

// Очищаем содержимое и она снова готова к использованию.

//void clear();

// Вернуть элемент с заданным ключём:

// Обязательно предполагается что ключ существует внутри таблицы.

//Item get(integer key);

//Item readmax();

//integer readkeymaxelm();

// Вставить элемент item в очередь по

// приоритетам если элемент item имеет ключ key.

//template <class Item>

//void insert(Item item, integer key);

// Возвратить максимальный элемент

// и удалить его.

//Item getmax();

// Заменяет элемент с ключём key на элемент val с тем же ключём key.

// При этом ключ key должен быть уникальным.

//void modify(integer key, Item val);

// Удаление элемента с заданным значением ключа.

//void remove(integer key);

// У элемента изменть значение старого ключа на новый ключ

// при этом меняется и само содержимое элемента.

//void change(integer key\_serch, integer key\_new, integer item\_new);

// Ключи должны быть уникальны, целочисленны и различны.

// Двух одинаковых ключей быть недолжно, иначе коллизия в хеш таблице.

template <class Item>

class PQ

{

private:

// Хранение binary heap.

Item \*pq;

// Обратный доступ по номеру в qp на ячейку в hash.

integer \*qp; // Ссылка на хеш таблицу.

// Доступ по ключу к полю в pq.

integer \*hash; // Хеш таблица !!!

integer N;

integer isize;

integer ihash\_size;

public:

PQ(integer maxN, integer max\_key\_size)

{

isize = maxN;

pq = new Item[maxN + 1];

qp = new integer[maxN + 1];

for (integer i\_1 = 0; i\_1 < maxN + 1; i\_1++) {

// Инициализация: таблица пуста т.к. поле 0

// в массиве pq никогда не используется.

qp[i\_1] = 0;

}

N = 0;

// Хеш таблица !!!

ihash\_size = max\_key\_size;

hash = new integer[max\_key\_size+2];

for (integer i\_1 = 0; i\_1 < max\_key\_size + 2; i\_1++) {

// Инициализация: таблица пуста т.к. поле 0

// в массиве pq никогда не используется.

hash[i\_1] = 0;

}

}

~PQ()

{

if (pq != NULL) delete[] pq;

N = 0;

if (qp != NULL) delete[] qp;

if (hash != NULL) delete[] hash;

}

void print\_log(char ch) {

printf("%c\n",ch);

for (integer i\_1 = 1; i\_1 <= N; i\_1++) {

#if doubleintprecision == 1

printf("[%lld %lld] ",pq[i\_1],qp[i\_1]);

#else

printf("[%d %d] ", pq[i\_1], qp[i\_1]);

#endif

}

printf("\n");

system("PAUSE");

}

// Меняет местами значения элементов с ключами i и j.

// Сохраняет порядок кучи.

void exchange\_speshial\_for\_Saad(integer i, integer j) {

//Item t;

//t = pq[hash[j]];

//pq[hash[j]] = pq[hash[i]];

//pq[hash[i]] = t;

// Этот обмен местами сохраняет порядок кучи.

Item t1 = get(i);

//Item t2 = get(j);

//this->remove(i);

//this->remove(j);

//this->insert(t1, j);

//this->insert(t2, i);

// быстродействущая модификация

this->remove(j);

this->insert(t1, j);

}

// Очищаем содержимое и она снова готова к использованию.

void clear()

{

for (integer i\_1 = 0; i\_1 < N + 1; i\_1++) {

// Ускоренная очистка хеш таблицы.

hash[qp[i\_1]] = 0;

}

for (integer i\_1 = 0; i\_1 < isize + 1; i\_1++) {

// Инициализация: таблица пуста т.к. поле 0

// в массиве pq никогда не используется.

qp[i\_1] = 0;

}

N = 0;

/\*

for (integer i\_1 = 0; i\_1 < ihash\_size + 2; i\_1++) {

// Инициализация: таблица пуста т.к. поле 0

// в массиве pq никогда не используется.

hash[i\_1] = 0;

}

\*/

}

bool empty() const

{

return N == 0;

}

// Есть ли элемент с данным ключём в таблице ?

bool isfound(integer key) {

if (hash[key] == 0) {

// Элемент отсутствует в хеш таблице.

return false;

}

return true;

}

// Вернуть элемент с заданным ключём:

// Обязательно предполагается что ключ существует внутри таблицы.

Item get(integer key) {

if (hash[key] == 0) {

// Элемент отсутствует в хеш таблице.

printf("priority queue get ERROR: get element not found.\n");

system("pause");

exit(1);

}

return pq[hash[key]];

}

// Просто прочитать масимальный элемент.

Item readmax()

{

return pq[1];

}

integer readkeymaxelm() {

return qp[1];

}

// Вставить элемент item в очередь по

// приоритетам если элемент item имеет ключ key.

template <class Item>

void insert(Item item, integer key)

{

if (N + 1 > isize) {

printf("ERROR!!! priority\_queue stack overflow...\n");

#if doubleintprecision == 1

printf("N=%lld\n",N);

#else

printf("N=%d\n", N);

#endif

system("pause");

exit(1);

}

else {

pq[++N] = item;

hash[key] = N;

qp[N] = key;

fixUp(pq, qp, hash, N);

}

//print\_log('i');

}

// Возвратить максимальный элемент

// и удалить его.

Item getmax()

{

exch(1, N, pq, qp, hash);

fixDown(pq, qp, hash, 1, N - 1);

return pq[N--];

}

// Заменяет элемент с ключём key на элемент val с тем же ключём key.

// При этом ключ key должен быть уникальным.

void modify(integer key, Item val)

{

if (hash[key] == 0) {

// Элемент отсутствует в хеш таблице.

printf("priority queue modify ERROR: get element not found.\n");

system("pause");

exit(1);

}

pq[hash[key]] = val;

// Теперь необходимо восстановить порядок кучи.

integer i = hash[key];

fixUp(pq, qp, hash, i);

fixDown(pq, qp, hash, i, N);

}

// Удаление элемента с заданным значением ключа.

void remove(integer key)

{

if (N > 0) {

if (hash[key] == 0) {

// Элемент отсутствует в хеш таблице.

// Ничего не делаем т.к. элемента уже нет.

}

else {

// Удаление.

if (hash[key] == N) {

N--;

hash[key] = 0;

qp[N + 1] = 0;

// Ключ исключён из таблицы.

}

else {

integer i = hash[key];

exch(hash[key], N, pq, qp, hash);

hash[qp[N]] = 0;

qp[N] = 0;

N--;

// Теперь необходимо восстановить порядок кучи.

fixUp(pq, qp, hash, i);

fixDown(pq, qp, hash, i, N);

}

}

}

//print\_log('r');

}

// У элемента изменть значение старого ключа на новый ключ

// при этом меняется и само содержимое элемента.

void change(integer key\_serch, integer key\_new, integer item\_new)

{

if (hash[key\_serch] == 0) {

// Элемент отсутствует в хеш таблице.

if (hash[key\_new] != 0) {

// Элемент присутствует в хеш таблице.

pq[hash[key\_new]] = item\_new;

// Теперь необходимо восстановить порядок кучи.

fixUp(pq, qp, hash, hash[key\_new]);

fixDown(pq, qp, hash, hash[key\_new], N);

}

else {

// Вставка нового ключа с новыми данными.

insert(item\_new, key\_new);

}

}

else {

if (hash[key\_new] != 0) {

// удаление старого ключа со всем его содержимым.

remove(key\_search);

// Элемент присутствует в хеш таблице.

pq[hash[key\_new]] = item\_new;

// Теперь необходимо восстановить порядок кучи.

fixUp(pq, qp, hash[key\_new]);

fixDown(pq, qp, hash[key\_new], N);

}

else {

// key\_new отсутствует.

hash[key\_new] = hash[key\_search];

hash[key\_search] = 0; // исключение из дерева.

pq[hash[key\_new]] = item\_new;

qp[hash[key\_new]] = key\_new;

// Теперь необходимо восстановить порядок кучи.

fixUp(pq, qp, hash, hash[key\_new]);

fixDown(pq, qp, hash, hash[key\_new], N);

}

}

}

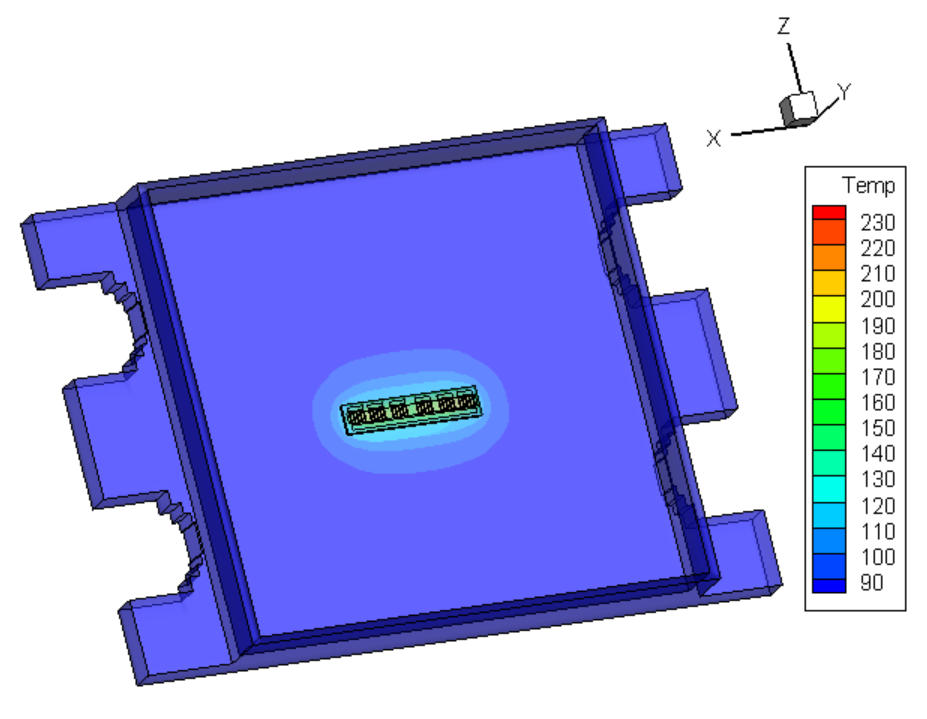
};

// Все операции O(1) и только операция delete log2(N).

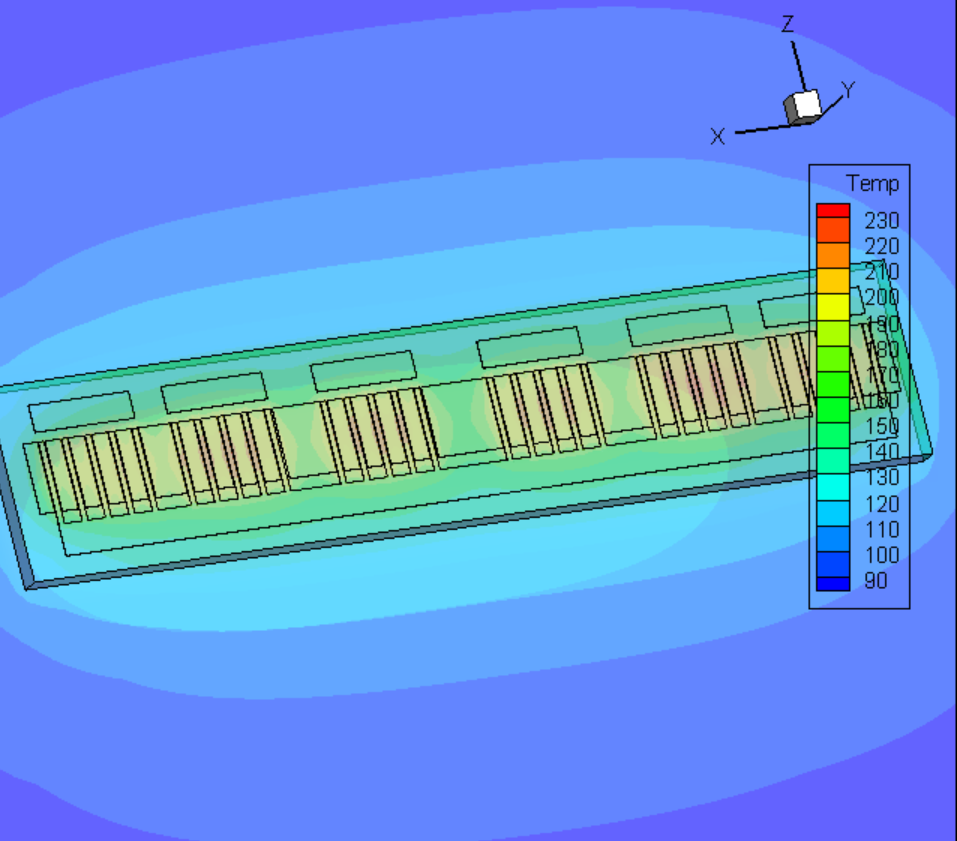
**Результаты расчётов**.

Таблица 1.

|  |  |  |  |
| --- | --- | --- | --- |
| Задача | Время, с | | |
| Линейный поиск | Двоичная куча и хеш таблица | И то и другое совместно |
| Задача №1 | 14мин 36с (5мин 30с) | 10мин 50с (1мин 45с) | 16 мин 22с (7мин 15с) |
|  |  |  |  |



а).



б).

Рис. 1. Поле температур в задаче 1, °С.

**Выводы**.

Предложен работающий программный код на языке С++ предлагающий решение проблемы быстродействия нахождения ilu(p) разложения в условиях работы BiCGStab+CAMG алгоритма.

Предлагаемое использование структуры данных двоичной кучи и хеш таблицы действительно быстрее чем простой линейный поиск, что продемонстрировано в Таблице 1.

**Литература**

[1]. Ю.Саад Итерационные методы. М. МГУ, 2013.

[2]. Роберт Седжвик Фундаментальные алгоритмы на С++.