Билеты по инфе 2 сем

1 билет

1.1 Ссылочный тип данных

В языках со строгой дисциплиной описаний (Паскаль) динамический объект не может иметь собственного имени, так как все идентификаторы языка должны быть описаны, поэтому принято не именовать, а обозначать динамический объект посредством ссылки на него. Переменная-ссылка должна быть описана в разделе объявлений программы как переменная ссылочного типа. При этом сама ссылка является статическим объектом. Ссылка занимает всего лишь одно машинное слово, что совсем немного (0(1)) по сравнению с возможным размером растущего динамического объекта. Ссылочный тип - это такой же простой скалярный тип, как целый, вещественный и логический, также имеющий прямую аппаратную поддержку. Элементами множества значений этого типа являются конкретные ссылки на объекты указанного типа, созданные в основной памяти в процессе выполнения программы. Разные указанные типы порождают разные ссылочные типы, множества значений которых не пересекаются. Однако пустое ссылочное значение nil принадлежит любому из ссылочных типов.

Для переменных в точности одного и того же ссылочного типа определены операции присваивания и разыменования и отношение равенства. Операция разыменования обеспечивает доступ к значению обозначаемого ссылкой объекта. В Си обозначается звездочкой слева от указателя (*p). Операция разыменования имеет аппаратную поддержку в любом современном процессоре (косвенная адресация). «Важно делать различие между переменной-указателем и указуемым объектом и быть очень осторожным при присваивании и сравнении указателей».

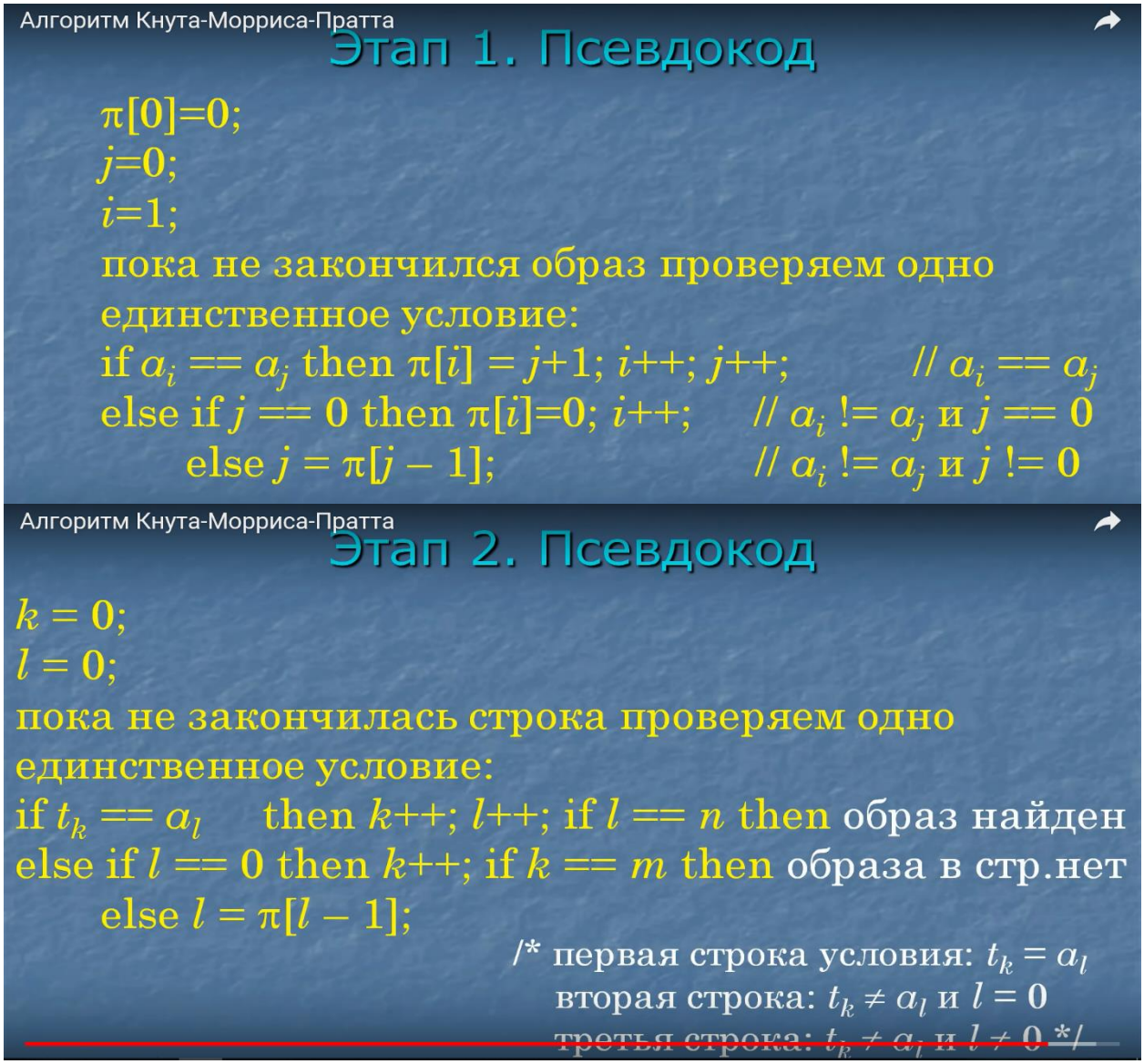
Динамические объекты определенного типа порождаются встроенными процедурами Паскаля и Си (new и malloc). В C++ порождение динамических объектов выполняется оператором языка new, который помимо выделения памяти и собственно порождения объекта вызывает конструктор объекта, выполняющий его инициализацию. При работе процедуры new память для динамических объектов выделяется из так называемой «кучи» - области основной памяти машины, зарезервированной для этой цели. В языке Паскаль определена также специальная процедура dispose, уничтожающая динамический объект, ссылка на который передана ей в качестве фактического параметра.

Если значение ссылочной переменной утеряно, то связь с соответствующим объектом безвозвратно теряется. После уничтожения объекта все ссылки на него становятся некорректными.

1.2 Алгоритм Кнута-Морриса-Пратта

Алгоритм, требующий N сравнений, вместо N*M. Алгоритм основывается на том, что мы не проходим заново уже пройденную часть, а продвигаемся по тексту, учитывая значение пройденной строки. При каждом несовпадении двух знаков образец сдвигается по последовательности вперед на все пройденное расстояние, поскольку при анализе образца установлено, что меньшие сдвиги не могут привести к полному совпадению.

Псевдокод взят из интернета:



Временная сложность алгоритма КМП: O(N+M), где N – длина, а M – длина строки, это намного лучше чем сложность прямого поиска, где сложность составляет O(N*M).

2 билет

2.1 Деревья выражений

В синтаксическом анализе определяют формальные грамматики, которые представляют собой множество правил подстановки (как в алгоритмах Маркова). Так, выражение можно определить как множество термов, разделяемых знаками «+» или « - »; в свою очередь, терм есть совокупность множителей, разделяемых знаками «*» или «/»; наконец, множитель может быть либо идентификатором, либо константой. Иерархическая структура выражения может быть описана древовидной схемой.

Обходя дерево выражения разными способами, мы получим три различные очереди вершин, которые представляют собой ни что иное, как широко используемые в информатике различные формы записи выражений: префиксную, привычную инфиксную, но без скобок, задающих порядок выполнения операций, и постфиксную (обратную польскую).

2.2 Таблицы с прямым доступом (hesh таблицы)

Поместим нашу таблицу в обычный массив и построим преобразование ключей в адреса в памяти. При поиске в таблице, как в случае поиска подстроки алгоритмом Рабина-Карпа, необходимо проверять точное совпадение ключей. Метод прямой адресации таблиц на основании функции ключа получил название хеширование (hashing).

Требования к хэш-функции: преобразование ключей должно распределять их по всему диапазону значений и функция должна быть эффективно вычислима при помощи небольшого числа аппаратных операций.

Эффективный способ вычисления функции расстановки - применение битовых аппаратно-поддерживаемых операций к части ключа в двоичном представлении. Правда, этот эффективный способ нередко проигрывает в равномерности.

Если при поиске в хеш-таблице обнаружилось, что строка, соответствующая заданному ключу, не содержит искомого элемента (с этим же ключом), то необходима вторая попытка переадресовать этот ключ внутри таблицы. Эта переадресация должна быть такой же быстрой, как первичное хеширование. Существует несколько методов генерации вторичного индекса:

1. организовать список строк с идентичным первичным ключом II(к). Элементы этого списка размещаются либо в основной таблице, либо вне ее в области переполнения. Недостаток этого способа в том, что необходимо обеспечивать управление этими вторичными списками с их хоть и малым, но линейным временем поиска
2. искать желаемый элемент или пустую строку в окрестности этого ключа (т. н. открытая адресация). Во второй попытке последовательность индексов должна однозначно вырабатываться из любого заданного ключа
3. случайное. К вычисленному индексу добавляется случайная величина, масштабированная под длину таблицы. Главное в таком методе - суметь восстановить случайную последовательность при поиске в таблице

3 билет

3.1 Вектор. Функциональная спецификация, логическое описание, физическое представление.

Вектор

Статистические массивы удобны размещением в памяти и обращениями к ним, но не всегда заранее известно сколько памяти нужно выделить. В этой ситуации используются динамические массивы или векторы. Эти массивы располагаются в куче, и их размер может быть задан произвольно. Недостатком таких массивов является большее среднее время доступа к элементу. Это связано с иерархичностью памяти современных ЭВМ. Главным источником критики векторов является обязанность программиста вручную распределять память, что может служить источником ошибок. Но эта критика основывается на доводах 20-30-летней давности.

Функциональная спецификация

Вектор является последовательностью переменной длины, и время доступа к элементам этой последовательности постоянно и не зависит от длины последовательности. Количество элементов вектора не фиксировано и всегда может быть изменено. Вектор с 0 компонент называется пустым.
Как правило, операционные системы, выделяя блок памяти, возвращают адрес начала этого блока. Для доступа к компоненте вектора необходимо прибавить к адресу блока размер одной компоненты, умноженной на ее индекс.
Теперь определим поведение операции «изменение размера». При уменьшении размера вектора с n до m все компоненты с индексами 0,..., m-1 должны сохранить свое значение. При увеличении длины вектора с n до p компоненты с индексами 0,..., n - 1 должны сохранить свое значение, а с индексами n,... ,p - 1 могут быть или неопределены, или проинициализированы значениями по умолчанию.

Логическое описание и физическое представление

Сначала опишем сам тип вектор. Он должен включать в себя описатель хранимой последовательности и ее длину.

```
typedef struct
{
    T* data;
    int size;
} Vector;
```

В процессе создания вектора определяется его длина:

```
void Create(Vector * v, int sz)
{
    v->size = sz;
    v->data = malloc(sizeof(T) * v->size);
}
```

Проверка на пустоту определяется путем сравнения длины с 0.
Как отмечалось раньше, длина является важной характеристикой вектора. Ее получение является простой операцией, выполняемой за постоянное время.

```
int Size(Vector* v)
{
    return v->size;
}
```

Теперь определим операции чтения и записи компоненты по данному индексу.

```
bool Load(Vector* v, int &i)
{
    if((i >= 0) && (i < v->size))
    {
        i = v->data[i];
        return true;
    }
    return false;
}
void Save(Vector* v, int i, T t)
{
    if(( i >= 0) && (i < v->size))
        v->data[i] = t;
}
```

Для написания функции изменения размера не придется писать копирование, поскольку библиотечные функции SetLength и realloc выполняют необходимые действия.

```
void Resize(Vector* v, int sz)
{
    v->size = sz;
    v->data = realloc(v->data, sizeof(T) * v->size);
}
```

Отношение равенства векторов играет большую роль в математике, поэтому эта операция просто должна быть определена. Два вектора считаются равными тогда и только тогда, когда равны их длины, а также равны соответствующие компоненты.

```
bool Equal(Vector* l, Vector* r)
{
    if(l->size != r->size)
        return false;
    for(int i = 0; i < l->size; i++)
        if(l->data[i] != r->data[i])
            return false;
    return true;
}
```

По окончании работы необходимо удалить вектор и вернуть память ОС.

```
void Destroy(Vector* v)
{
    v->size = 0;
    free(v->data);
}
```

3.2 Алгоритм Рутисхаузера

Один из наиболее ранних алгоритмов. Предполагает полную скобочную структуру, при этом неявное старшинство операций не учитывается.
Обрабатывая выражение с полной скобочной структурой, алгоритм присваивает каждому символу номер уровня по правилу:

- если это открывающаяся скобка или переменная, то значение уровня увеличивается на 1
- если это знак операции или закрывающая скобка, то уровень уменьшается на 1

Для выражения (A+(B*C)) значения уровней таковы:

Символ	(	A	+	(	B	*	C	)	)
Номер уровня	1	2	1	2	3	2	3	2	1

Основные этапы алгоритма:

1. расставить уровни
2. отыскать элемент с максимальным уровнем
3. выделить тройку - два операнда с максимальным значением уровня и операцию между ними
4. результат выделенной операции обозначить вспомогательной переменной
5. из исходной строки удалить выделенную тройку с ее скобками, а на ее место поставить вспомогательную переменную, с уровнем на единицу меньше, чем у тройки
6. выполнять 2-5, пока в строке не останется одна переменная

4 билет

4.1 Стек. Функциональная спецификация

Стек

Принцип: первый пришел, последний ушел (стопка книг).
Стек - структура с последовательным доступом, неразрушающей записью и разрушающим чтением.
Операции:

- Пополнение
- Проверка на пустоту
- Просмотр верхнего (самого нового) элемента

- Уничтожение последнего добавленного

Стеки используются для:

- Изучения физических процессов
- Синтаксического анализа текста
- Выполнения рекурсивных процедур
- Некоторые ЭВМ, микропроцессоры и языки программирования имеют стековую структуру

Функциональная спецификация

Тип ST или стек объектов типа T, характеризуется операциями:

- СОЗДАТЬ: Ø → ST
- ПУСТО: ST→ boolean
- ДЛИНА: ST → N
- В_СТЕК: STx T→ ST
- ИЗ_СТЕКА: ST→ST
- ВЕРХ:ST →T
- УНИЧТОЖИТЬ: ST→ Ø

Свойства:

1. ПУСТО(СОЗДАТЬ) = true
2. ПУСТО(В_СТЕК(s,t)) = false
3. ВЕРХ(В_СТЕК(s,t)) = t

4.2 Дерево поиска (или сбалансированные деревья поиска см. билет 13)

Двоичные деревья часто употребляются для представления множеств данных, элементы которых должны быть найдены по некоторому ключу (полю данных). Если дерево организовано так, что для каждой вершины ti справедливо утверждение, что все ключи левого поддерева tj меньше ключа Ц, а все ключи правого поддерева Ц больше его, то такое дерево называют деревом поиска. По построению дерева поиска, переходя к одному из поддеревьев, мы автоматически исключаем из рассмотрения другое поддерево, содержащее половину узлов.

```
tree locate(int x, tree t)
{
    while(t && t->key != x)
        if(t->key < x)
            t = t->r;
        else
            t = t->l;
    return t;
}
```

Поиск по дереву с включениями

Требуется определить частоту вхождения каждого из слов в последовательность, поступающую из входного текстового файла. Надо либо найти слово в дереве и добавить 1 к находящемуся в соответствующем узле счётчику его вхождений, либо, в случае

неуспеха, включить новый элемент в дерево с сохранением нелинейного иерархического порядка и поисковой структуры и установить для него единичное значение счётчика. Процедура поиска с включением search() сначала должна осуществлять поиск в дереве аналогично процедуре locate(). В случае неудачи поиск оканчивается на одном из листьев дерева, к которому привела попытка поиска. Поскольку все предыдущие проверки на пути к этому листу пройдены, то мы имеем готовый маршрут поиска нового элемента, а данный лист представляет собой искомое место для вставки этого элемента в поисковое дерево.

```
struct word;
typedef word* wp;
struct word
{
    int key;
    int count;
    wp l, r;
}
void search(int x, wp &p)
{
    if(!p)
    {
        p = malloc(sizeof(wp));
        p->key = x;
        p->count = 1;
        p->l = p->r = 0;
    }
    else
        if x < p->key
            search(x, p->l);
        else
            if(x > p->key)
                search(x, p->r);
            else
                p->count++;
}
```

5 билет

5.1 Линейный список. Физическое представление. Итераторы.

Для реализации сложных динамических структур данных цепного или сплошного характера принято использовать так называемые итераторы.
Для удобства организации списка и обеспечения единообразия доступа к нему определим объекты, обладающие функциями перехода от данного элемента списка к соседним. Зададим для них отношения равенства и неравенства. Два таких объекта равны тогда и только тогда, когда они указывают на один и тот же элемент списка. Также предоставим возможность чтения и записи элемента списка посредством введенных объектов. Такие объекты принято называть итераторами, и, конечно же, для них надо определить соответствующий тип данных.
Структура элемента списка:

```
struct Item
{
	struct Item* prev;
	struct Item* next;
	T data;
};
typedef struct
{
	Item* node;
} Iterator;
```

Равенство:

```
bool Equal(const Iterator* lhs, const Iterator* rhs)
{
	return lhs->node == rhs->node;
}
```

Переход на следующий элемент:

```
Iterator Next(Iterator* i)
{
	i->node = i->node->next;
	return i;
}
```

Переход на предыдущий элемент:

```
Iterator Prev(Iterator* i)
{
	i->node = i->node->prev;
	return i;
}
```

Чтение текущего элемента:

```
T Fetch(const Iterator* i)
{
```

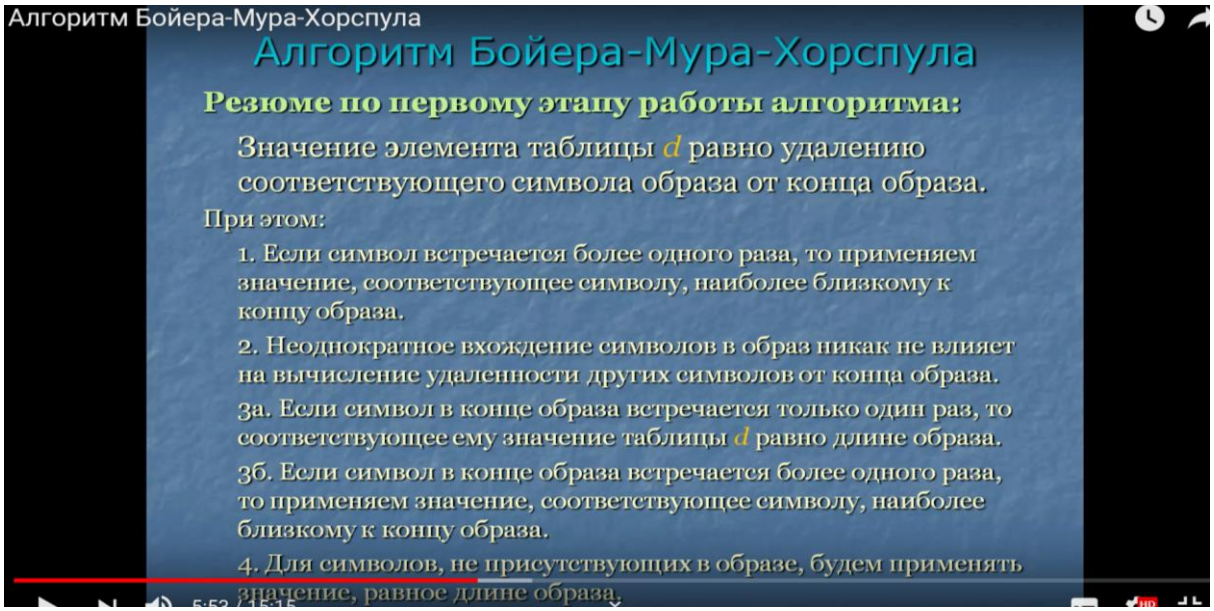
```
      return i->node->data;
}
void Store(const Iterator* i, const T t)
{
    i->node->data = t;
}
```

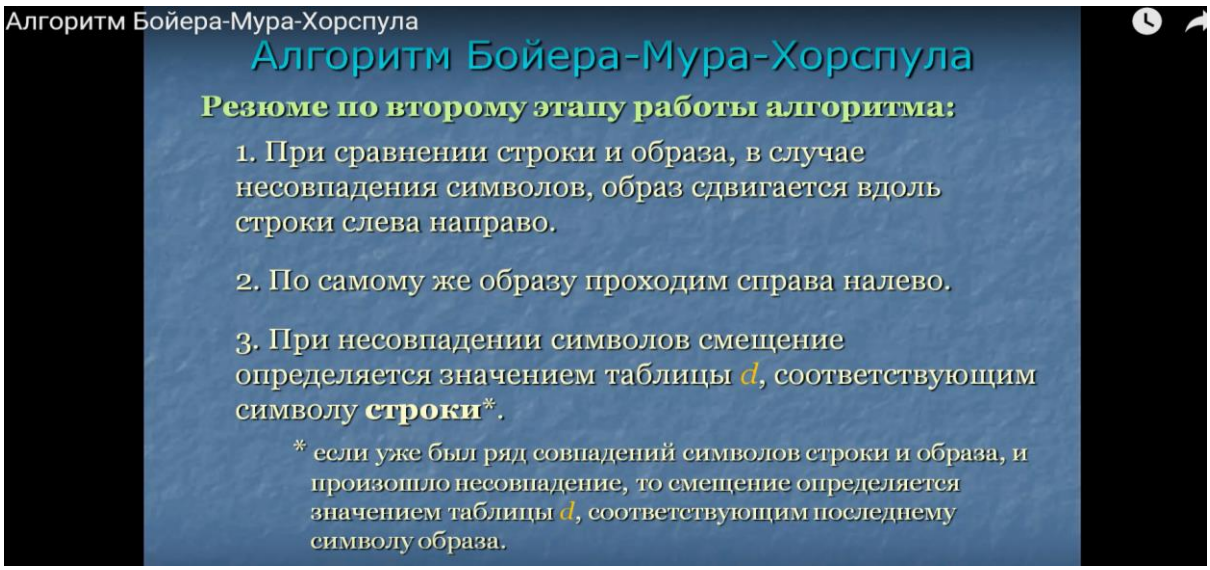
Во всех функция отсутствуют проверки, потому что они загромождают код, поэтому все проверки должны быть осуществлены пользователем.

5.2 Алгоритм Бойера-Мура (Боейра-Мура-Хорспула)

Остроумная схема КМП-поиска дает подлинный выигрыш только тогда, когда неудаче предшествовало некоторое число совпадений. Лишь в этом случае образец сдвигается более чем на единицу. В 1977 году Р. Бойер и Д. Мур предложили другой метод поиска, который не только улучшает обработку самого плохого случая (когда и в образце, и в последовательности многократно повторяется небольшое число букв), но и дает выигрыш в промежуточных ситуациях.
БМ-поиск основан на необычном соображении - сравнение литер проводится не с начала образца, слева направо, а наоборот, с конца образца справа налево. Пусть для каждого знака алфавита x dx означает его расстояние от самого правого вхождения в образец до его конца. Если в процессе поиска обнаружено расхождение образца и строки, то образец сразу же можно сдвинуть вправо на d_Pm_1 позиций, т. е. на число позиций, скорее всего, большее единицы. Если Pm-1 в образце вообще не встречается, то сдвиг можно сделать сразу на всю длину образца.

Описание алгоритма из интернета:





Код алгоритма:

```
i = M;
j = M;
while(j > 0 && i < N)
{
    //Q(i-M)
    i = M;
    k = i;
    while(j > 0 && s[k-1] == p[j-1])
    {
        //P(k-j,j) & (k-i == j-M
        k--;
        j--;
    }
    i += d[s[i-1]];
}
```

6 билет

6.1 Двоичное дерево. Физическое представление. Прошивка.

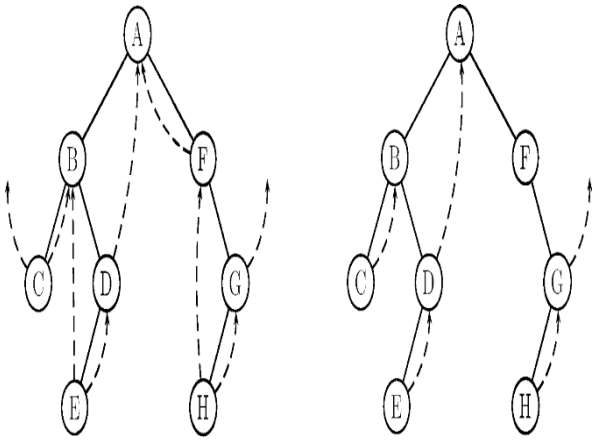
Физическое представление (массив)

Введём жёсткое размещение элементов дерева в массиве. В первом элементе массива разместим корень дерева, во 2-ом и 3-ем - его левого и правого потомков. Далее поместим пары потомков потомков и т. д. Сыновья элемента дерева с индексом i хранятся в элементах массива с индексами 2i и 2i + 1. Согласно данной схеме размещения, j-ый элемент i-ого уровня имеет индекс 2^(i-1) + j - 1.

Этот метод весьма экономичен по памяти. Основным неудобством сплошного представления дерева является высокая цена вставки и удаления элементов. Не мал и перерасход памяти на пустые элементы. Как всегда, за ликвидацией этих недостатков мы обратимся к динамическим структурам.
Для деревьев можно использовать рекурсивные ссылочные представления, которые были разработаны для списков, но с той лишь разницей, что указатели вперёд и назад по линейной структуре теперь направляются к левому и к правому поддеревьям соответственно. Тип для вершины дерева, конечно же, фиксирован, но имеет двойное рекурсивное разветвление.

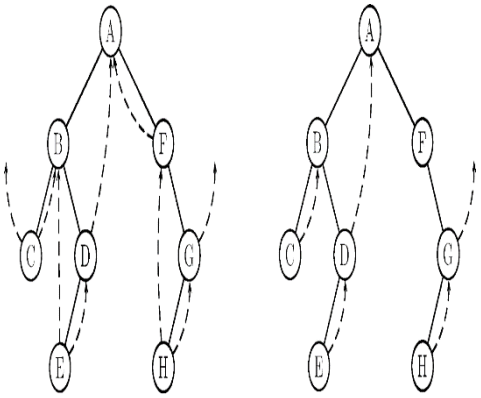
Прошивка

В прошитых бинарных деревьях вместо пустых указателей используются специальные связи-нити и каждый указатель в узле дерева дополняется однобитовым признаком ltag и rtag, соответственно. Признак определяет, содержится ли в соответствующем указателе обычная ссылка на поддерево или в нем содержится связь-нить.
Связь-нить в поле l указывает на узел - предшественник в обратном порядке обхода (inorder), а связь-нить в поле r указывает на узел - преемник данного узла в обратном порядке обхода.
Введение признаков ltag и rtag не приводит к сколько-нибудь значительному увеличению затрат памяти, зато упрощает алгоритм обхода деревьев, так как для прошитых деревьев можно выполнить нерекурсивный обход без использования стека.
Пример прошитого дерева:

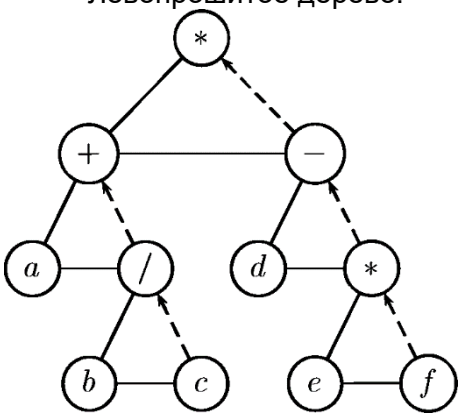


Также существуют правопршитые (левопрошитые), в которых есть только rtag (ltag), для экономии памяти.

Правопршитое дерево:



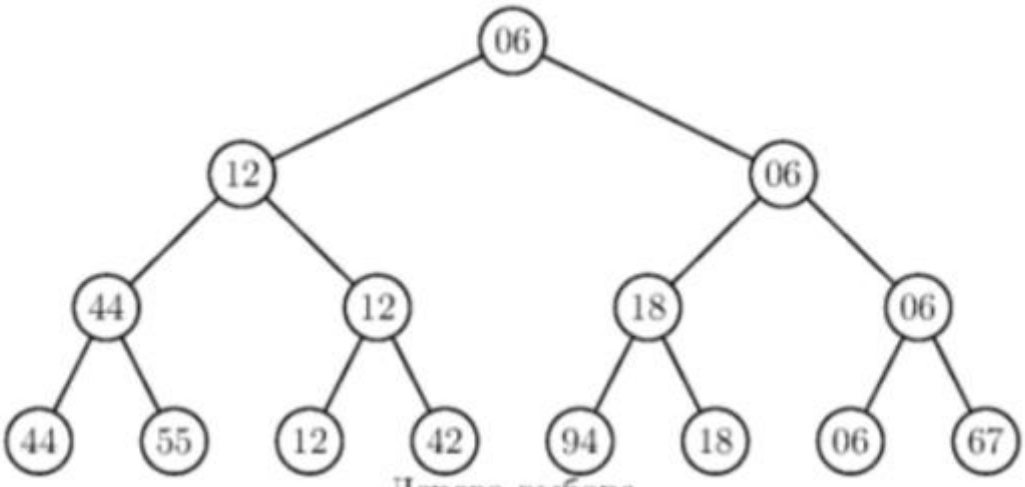
Левопрошитое дерево:



6.2 Турнирные сортировки (пирамидальная сортировка)

В сортируемом множестве можно сравнить пары соседних элементов → из **n/2** сравнений получаем меньшую по значению ключей половину (победители); также из получаем четверть и т.д., пока не получим один элемент.

Результат – дерево выбора (турнирная таблица – двоичное дерево, в котором для каждого двудутного узла один из сыновей – он сам (победитель), а другой больше своего отца) (в корне – самый маленький элемент)



Дерево выбора

Перевод в множество: **а)** возьмём корень и запишем его в множество

б) спускаемся от победителя к призёрам по пути победителя, опустошая вершины

в) поднимаясь по этому пути, выполняем перераспределение мест, как будто победителя из п.а не было (элемент, соревнующийся с пустым местом, автоматом переходит в следующий тур)

г) теперь место победителя занял новый элемент; переходим к п.1

Достоинство – сложность всегда **O(n * ln n).**

{для уменьшения расхода памяти необходимо применить пирамидальное улучшение традиционных древовидных сортировок (**hi =<h2ihi =<h[2i+1]);** в пирамиде нет повторов элементов сортируемого множества}

Строительство пирамиды: **1)** первоначально элемент помещается в вершину**, i = 0**

2) рассмотрим тройку элементов с индексами **i, 2i, 2i+1** (узел и его дети)

3) выберем наименьший из п.2: если он на позиции i, элемент вставлен; ELSE обозначим позицию наименьшего буквой **j, где j=j2i** или **j = 2i + 1**, меняем местами элементы i и j

4) переходим к п.2

Пирамида отличается от турнирного дерева отсутствием дубликатов элементов сортируемой последовательности.

7 билет

7.1 Динамические и статические данные

Свойства объекта, которые остаются неизменными при любом исполнении называются **статическими**. Их можно определить по тексту программы. Пример : тип объекта.
Свойства объекта, изменяемые во время работы программы называются **динамическими**. Пример : конкретное значение переменной.
Часто статические и динамические характеристики называют соответственно характеристиками периода компиляции и периода выполнения.
Это деление характеристик на статические и динамические иногда оказывается слишком черно-белым.
Одна крайняя позиция представлена концепцией неограниченного динамизма, когда по существу любая характеристика обрабатываемого объекта может быть изменена при выполнении программы. Такая концепция не исключает прогнозирования и контроля, но и не связывает их жестко со структурой текста программы.
Другая крайняя позиция выражена в стремлении затруднить программисту всякое изменение характеристик объектов. Вводя объект, надо объявить характеристики, которым должно соответствовать всякое его использование.
Если память выделяется (распределяется) в процессе трансляции и ее объем не меняется от начала до конца выполнения программы, то такой объект является статическим. Если же память выделяется во время выполнения программы и ее объем может меняться, то такой объект является динамическим.

7.2 Адресный тип. Полиморфизм с Си при помощи адресного типа.

Адресный тип имеет множеством значений диапазон допустимых адресов ЭВМ. Объекты адресного типа совместимы по присваиванию и сравнению с любыми ссылочными переменными, а формальные параметры - с любыми ссылочными фактическими. Кроме того, адресный тип естественно совместим с целым и к нему применимы арифметические операции. Адресный тип полностью устраняет контроль типов, выполняемый компилятором, и его следует употреблять только для разработки родовых модулей низкого уровня.
Для реализации полиморфных функций, зависящих от типа передаваемых аргументов, в C++ используется понятие шаблона функции. Шаблон - это некоторая подпрограмма, предъявляющая определенные требования к типу своих аргументов.
Пример:

```
template<class T>
void inline swap(T& lhs, T& rhs)
{
      T t = lhs;
      lhs = rhs;
      rhs = t;
}
```

Единственным требованием, предъявляемым к типу аргумента, является наличие у типа оператора присваивания.

8 билет

8.1 Линейный список. Функциональная спецификация.

Линейный список

Линейные списки являются обобщением ранее изученных последовательных структур с ограниченным доступом: файлов, очередей и стеков. Они позволяют представить последовательность элементов так, чтобы каждый элемент был бы доступен вне зависимости от положения.
Линейный список - это представление в ЭВМ конечного упорядоченного динамического множества элементов типа T. Точнее, это не множество, а мультимножество, т. к. в последовательности могут находиться элементы с одинаковыми значениями. Элементы этого множества линейно упорядочены, но порядок определяется не номерами (индексами), а относительным расположением элементов. Линейные списки естественно использовать всякий раз, когда встречаются упорядоченные множества переменного размера.
Наиболее простой пример линейных списков в информатике - списки переменных в описаниях:
int x, y, z;

Для поиска элемента в списке надо просматривать его с начала, сравнивая искомое значение с текущим значением элемента.
Для добавления нового элемента в список необходимо указать значение, перед (или после) которого надо сделать вставку.
Удаление элемента из списка обычно предваряется его поиском, занимающим в среднем N/2 шагов. Цена самого удаления обычно невелика и пропорциональна O(1).

Функциональная спецификация

Тип LT или двусвязный список объектов типа T, характеризуется операциями:

- СОЗДАТЬ: Ø → LT
- ПУСТО: LT→ boolean
- ДЛИНА: LT → N
- ПЕРВЫЙ:LT →T
- ПОСЛЕДНИЙ:LT →T
- СЛЕДУЮЩИЙ:LTx T→ LT
- ПРЕДЫДУЩИЙ:LTx T→ LT
- ВСТАВКА: LTx T x T→ LT
- УДАЛЕНИЕ: LTx T→ LT
- УНИЧТОЖИТЬ: LT→ Ø

Свойства:

1. ПРЕДЫДУЩИЙ(СЛЕДУЮЩИЙ(t)) = t
2. СЛЕДУЮЩИЙ(ПРЕДЫДУЩИЙ(t)) = t

8.2 Сортировка Хоара (быстрая сортировка)

Выберем наугад какой-нибудь промежуточный (разделяющий) элемент X (зерно) и будем просматривать сортируемый массив слева, пока не обнаружим элемент $a_i > x$, затем, проходя этот же массив справа, будем искать элемент $a_j < x$. Чтобы ликвидировать обнаруженную инверсию, поменяем местами эти два элемента (они в силу поиска с концов к середине находятся на довольно большом расстоянии) и продолжим процесс просмотра и перестановки, пока оба прохода не встретятся где-то в середине массива. Этот процесс классифицирует сортируемые элементы относительно зерна сортировки x: в нужной ли половине массива они находятся.

Рекурсивный вариант:

```
void QuickSort(int *a, int l, int r)
{
    int i, j, x, tmp;
    x = a[(l+r)/2];
    i = l;
    j = r;
    do
    {
```

```
        while(a[i] < x)
            i++;
        while(a[j] > x)
            j--;
        if( i < j)
        {
            tmp = a[i];
            a[i] = a[j];
            a[j] = tmp;
            i++;
            j--;
        }
    } while(i < j);
    if(l < j)
        QuickSort(a, l, j);
    if(i < r)
        QuickSort(a, i, r);
}
```

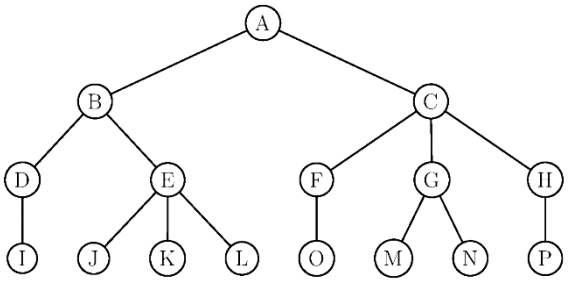
Среднее число перестановок: $M = n-1/n6$

9 билет

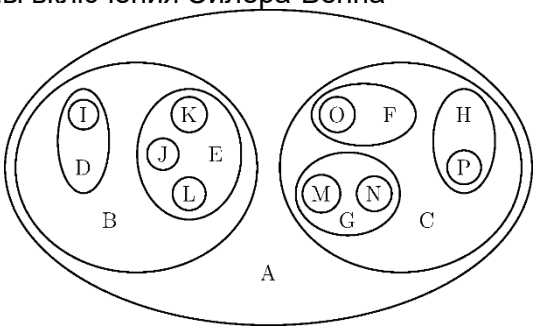
9.1 Представление и обработка деревьев общего вида.

Деревья - структура данных, наиболее приспособленная для решения задач искусственного интеллекта и синтаксического анализа.
Пусть T - некоторый тип данных. Деревом типа T называется структура, которая образована элементом типа T, называемым корнем дерева, и конечным, возможно пустым, множеством с переменным числом элементов - деревьев типа T, называемых поддеревьями этого дерева.
Это рекурсивное определение похоже на определения файла, очереди, стека и списка, но отличается от них нелинейностью ввиду наличия разветвлений.
Виды представления деревьев:

- Стандартное



- Вложенные диаграммы включения Эйлера-Венна



- иерархические скобочные структуры:

(A (B (D (I), E (J, K, L)), C (F (O), G (M, N),H (P))))

- Матрица смежности

Определения, относящиеся к деревьям:

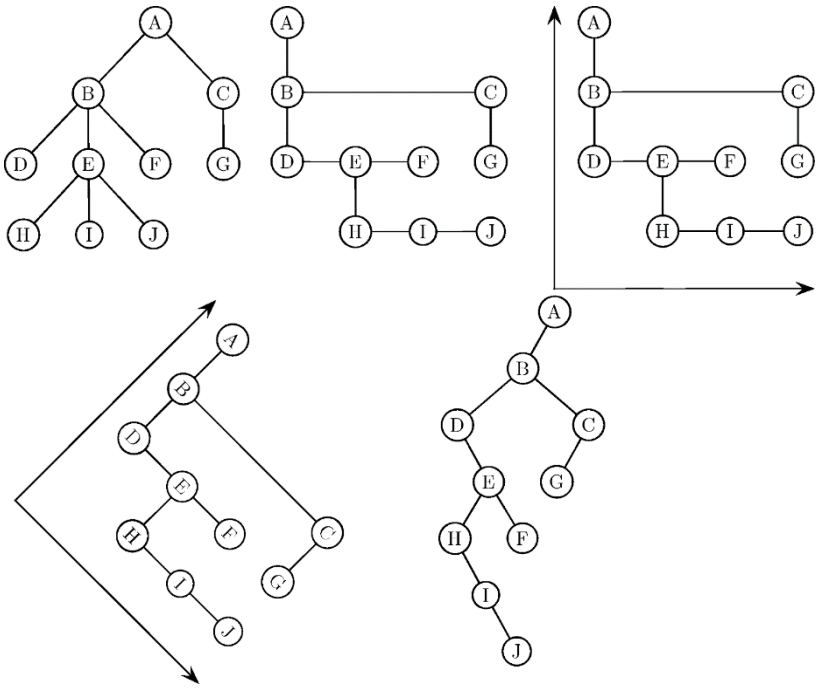
1. Элементы, входящие в дерево - узлы
2. Число поддеревьев узла - степень узла
3. Узел степени ноль - лист или концевой узел
4. Неконцевые узлы называются внутренними
5. Глубиной дерева называется наибольшее значение уровня

Для описания взаимного расположения узлов принята терминология генеалогических деревьев, соответствующая родственным отношениям между лицами преимущественно по мужской линии (отец, сын или предок и потомок, но дочерняя вершина).
Определенные нами таким образом деревья называются деревьями общего вида или сильно ветвящимися.

Двоичная интерпретация дерева общего вида

Для преобразования произвольного дерева в бинарное, надо в исходном дереве у каждого узла соединить его сыновей (от старшего к ближайшему младшему брату) и убрать все связи узлов с сыновьями, сохранив лишь связь с первым сыном, за которым выстраивается своеобразная очередь сыновей - нисходящий правосторонний каскад.
В общем виде алгоритм преобразования обычного дерева A в двоичное дерево B сводится к рекурсивному применению следующих правил:

- корень B есть корень A
- левое поддерево двоичного дерева B есть результат этого же преобразования первого поддерева дерева A, если оно существует
- правое поддерево двоичного дерева B есть результат этого преобразования очередного брата узла A, если он существует



Замечание: Изложенный рекурсивный алгоритм преобразования любого дерева общего вида в двоичное имеет и концептуальное значение. Подобно теореме Бойма-Джакопини-Миллса он конструктивно доказывает эквивалентность этих двух разновидностей деревьев: двоичные деревья есть деревья ограниченного разветвления аналогично тому, что схемы машин Тьюринга с ограниченным набором управляющих структур были полным эквивалентом диаграмм.

9.2 Алгоритм Рабина-Карпа

Взято из интернета:

Алгоритм Рабина-Карпа — это алгоритм поиска строки, который ищет шаблон, то есть подстроку, в тексте, используя хеширование. Этот алгоритм хорошо работает во многих практических случаях, но совершенно неэффективен, например, на поиске строки из 10 тысяч символов «a», за которыми следует «b», в строке из 10 миллионов символов «a». В этом случае он показывает своё худшее время исполнения $\underline{\Theta}(mn)$.

Вместо того, чтобы использовать более умный пропуск, алгоритм Рабина — Карпа пытается ускорить проверку эквивалентности образца с подстроками в тексте, используя хеш-функцию. Хеш-функция — это функция, преобразующая каждую строку в числовое значение, называемое *хеш-значением (хеш)*; например, мы можем иметь хеш от строки «hello» равным 5. Алгоритм использует тот факт, что если две строки одинаковы, то и их хеш-значения также одинаковы. Таким образом, всё что нам нужно, это посчитать хеш-значение искомой подстроки и затем найти подстроку с таким же хеш-значением.

Однако существуют две проблемы, связанные с этим. Первая состоит в том, что, так как существует очень много различных строк, между двумя различными строками может произойти коллизия — совпадение их хешей. В таких случаях необходимо посимвольно проверять совпадение самих подстрок, что занимает достаточно много времени, если данные подстроки имеют большую длину (эту проверку делать не нужно, если ваше приложение допускает ложные срабатывания). При использовании достаточно хороших

хеш-функций (смотрите далее) коллизии случаются крайне редко, и в результате среднее время поиска оказывается невелико.

Псевдокод из интернета:

```
int RabinKarp(string s[1..n], string sub[1..m])
hsub = hash(sub[1..m])
hs = hash(s[1..m])
for int i= 0; i < (n-m+1); i++)
if hs == hsub
if s[i..i+m-1] == sub
return i;
hs = hash(s[i+1..i+m])
return 0;
```

10 билет

10.1 Алгоритм Дейкстра

Способ разбора математических выражений, представленных в обычной инфиксной форме. Результат - обратная польская запись или дерево выражений. Алгоритм работает при помощи стека.

Пока не все символы (лексемы) обработаны:
- Прочитать лексему.
- Если лексема - число, добавить в очередь вывода.
- Если - функция, то поместить в стек.
- Если лексема - разделитель аргументов функции (например запятая):
  - пока символ на вершине стека не открывающая скобка, перекладывать операторы из стека в выходную очередь. Если в стеке не было открывающей скобки, то в выражении пропущен разделитель аргументов функции (запятая), либо пропущена открывающая скобка.
- Если лексема - оператор op1, то:
  - пока присутствует на вершине стека лексема оператор op2
    - переложить op2 из стека в выходную очередь
  - положить op1 в стек.
- Если - открывающая скобка, то положить его в стек.
- Если лексема - закрывающая скобка:
  - пока лексема на вершине стека не открывающая скобка, перекладывать
    - операторы из стека в выходную очередь.
    - выкинуть открывающую скобку из стека, но не добавлять в очередь вывода.
    - если на вершине стека лексема функции, добавить ее в выходную очередь.

если стек закончился до того, как был встречена открывающая скобка, то в выражении пропущена скобка.

Если больше не осталось лексем на входе:

пока есть операторы в стеке:

если лексема оператор на вершине стека - скобка, то в выражении присутствует незакрытая скобка.

переложить оператор из стека в выходную очередь.

Конец.

10.2 Сортировка слиянием

Еще одно усовершенствование быстрой сортировки связано с оценкой размера стека для хранения границ отложенных участков. В самом неблагоприятном случае, когда в стек заносятся одноэлементные участки, его размер оценивается в n элементов. Если же заносить в стек «более длинную часть» и продолжать разделение более короткой части, размер стека может быть ограничен log n. Тогда в программу нерекурсивной сортировки Хоара должны быть внесены следующие изменения:

```
if(j - l < t - i)
{
    if(i < r)
    {
        t->l = i;
        t->r = r;
        push(st, t);
    }
    r = j;
}
else
{
    if(l < j)
    {
        t->l = l;
        t->r = j;
        push(st, t);
    }
    l = i;
}
```

11 билет

11.1 Двоичное дерево. Функциональная спецификация.

Бинарное (двоичное) дерево - это конечное множество узлов, которое или пусто или состоит из корня и двух непересекающихся поддеревьев, называемых левым и правым поддеревьями данного узла.
Бинарное дерево не является частным случаем дерева общего вида, это особый вид структуры данных, хотя и близкий к дереву.
В бинарном дереве существенное значение имеет наклон ветвей, в то время как в обычном дереве он несущественен. Вторая особенность: бинарное дерево, в отличие от обычного, может быть пустым.
Бинарные деревья имеют важное значение в практике программирования; одна из причин этого заключается в том, что обычные деревья часто представляются с помощью специальных классов бинарных деревьев, т. к. их проще хранить и обрабатывать.

Функциональная спецификация

Тип BTT или двоичное дерево типа T, определяется так:

- СОЗДАТЬ: Ø → BTT
- ПОСТРОИТЬ: BTTx T x BTT→ BTT
- ПУСТО: BTT→ boolean
- КОРЕНЬ: BTT →T
- СЛЕВА: BTT→ BTT
- СПРАВА: BTT→ BTT
- УНИЧТОЖИТЬ: BTT→ Ø

Свойства:

1. ПУСТО(СОЗДАТЬ) = true
2. ПУСТО(ПОСТРОИТЬ(btl, t, btr)) = false
3. КОРЕНЬ(ПОСТРОИТЬ(btl, t, btr)) = t
4. СЛЕВА(ПОСТРОИТЬ(btl, t, btr)) = btl
5. СПРАВА(ПОСТРОИТЬ(btl, t, btr)) = btr
6. ПОСТРОИТЬ(СЛЕВА(bt), КОРЕНЬ(bt), СПРАВА(bt)) = bt

Операции над деревьями:

1. чтение данных из узла дерева;
2. создание дерева, состоящего из одного корневого узла;
3. построение дерева из заданных корня и нескольких поддеревьев;
4. присоединение к узлу нового поддерева;
5. замена поддерева на новое поддерево;
6. удаление поддерева;
7. получение узла, следующего за данным в определённом порядке.

11.2 Алгоритм Бауэра-Замельзона.

Ранний стековый метод. Используются два стека и таблица перехода. Один стек (T) используется при трансляции выражения, а второй (E) - во время интерпретации выражения. В таблице переходов задаются функции, которые должен выполнить при разборе выражения:

- f1: заслать операцию из входной строки в стек T, читать следующий элемент
- f2: выделить тройку - взять операцию с вершины T и два операнда с вершины E, заслать результат в T, читать следующую лексему строки
- f3: исключить лексему из T, читать следующий символ
- f4: выделить тройку – взять операцию с вершины T и два операнда с вершины E, результат занести в E, по таблице определить функцию для данной лексемы входной строки
- f5: выдача сообщения об ошибке
- f6: завершение работы

Таблица переходов ($ - пустой стек или строка):

		$	(	+	-	*	/	)
Операция на вершине стека T	$	6	1	1	1	1	1	5
	(	5	1	1	1	1	1	3
	+	4	1	2	2	1	1	4
	-	4	1	2	2	1	1	4
	*	4	1	4	4	2	2	4
	/	4	1	4	4	2	2	4

12 билет

12.1 Дек. Сравнительное описание. Примеры задач.

Дек - двухсторонняя очередь, динамическая структура данных, в которой элементы можно добавлять и удалять как в начало, так и в конец.
Операции над деком:

- включение элемента справа и слева
- исключение элемента справа и слева
- определение размера
- очистка

Пример задачи: Птицы, сидевшие на проводе, и разозлившиеся от поднятой пыли начали бежать по проводу в разные стороны (право и лево). Птицы бегут с одинаковой скоростью, а встретившись, начинают бежать в противоположном направлении. Если птица добегает до конца провода, то она взлетает. Необходимо написать программу, которая при данных длине провода, начальным позициям и направлениям бега выяснит, в какой момент времени каждая улетит с провода.

Решение: Введение двух деков. При взлете птицы из дека удаляется крайний элемент, а сами деки меняются местами. Храня пары <дек; сколько надо добавить к числам в нем>, можно находить через какое время и в какую сторону улетит очередная птица. Поскольку порядок птиц не меняется, дополнительно следует хранить сортированный массив координат начальных положений птиц, чтобы определить, какая из них покинула провод.

12.2 Алгоритмы обхода деревьев

Для бинарного дерева определены три способа обхода: прямой (или сверху вниз), обратный (или слева направо) и концевой (или снизу вверх). Эти способы определяются рекурсивно.

1. При прямом обходе
    a. если дерево пусто, конец
    b. берется корень
    c. выполняется обход левого поддерева
    d. выполняется обход правого поддерева
2. При обратном обходе

. если дерево пусто, конец
a. выполняется обход левого поддерева
b. берется корень
c. выполняется обход правого поддерева

3. При концевом обходе

. если дерево пусто, конец
a. выполняется обход левого поддерева
b. выполняется обход правого поддерева
c. берется корень

Обход дерева общего вида:

1. Поиск в глубину:

a. если дерево пусто, конец
b. берется корень
c. выполняется поиск в глубину по старшему сыну
d. выполняется поиск в глубину по следующему брату

2. Поиск в ширину:

. поместить в пустую очередь корень дерева
a. если очередь узлов пуста, конец
b. извлечь первый элемент из очереди узлов и поместить в ее конец всех сыновей по старшинству
c. повторить поиск с пункта 2

13 билет

13.1 Файл. Функциональная спецификация

Файл

Динамические объекты могут размещаться не только в основной, но и во внешней памяти ЭВМ. Файлы могут быть внешними (долгосрочное хранение информации) и внутренними (память для них выделена временно).

Функциональная спецификация

Обозначим через FT файловый тип с компонентами типа T.
Множество значений файлового типа строго определяется следующими правилами:

1. {} есть файл типа FT(пустая последовательность или пустой файл);
2. если f есть файл FT и t есть объект типа t, то f || {t} есть файл типа FT (|| - конкатенация);
3. никакие другие значения не являются файлами типа FT;

Базовое множество атрибутов файлового типа:

1. операция конкатенации;
2. операция присваивания;
3. отношения равенства;
4. функции: создание, доступ, модификация, уничтожение.

13.2 Дерево поиска (Скорее всего сбалансированные деревья поиска, иначе см. билет 4)

Сбалансированным деревом называется такое дерево, высоты поддеревьев каждой из вершин которого отличаются не более, чем на единицу. Такие деревья названы AVL-деревьями.
В сбалансированных деревьях за время, пропорциональное O(log n) можно:

- найти вершину с данным ключом
- включить новую вершину с заданным ключом
- исключить вершину с заданным ключом

Высота сбалансированного дерева с n вершинами hb(n) находится в пределах
log(n + 1) < h(n) < 1.4404 • log(n + 2) - 0.328
Определение деревьев Фибоначчи:

1. Пустое дерево есть дерево Фибоначчи высоты 0
2. Единственная вершина есть дерево Фибоначчи высоты h
3. Если T1 и T2 - деревья Фибоначчи высоты h-1 и h-2, то Th = <T1, h, T2> также дерево Фибоначчи высоты h
4. Никакие другие деревья деревьями Фибоначчи не являются

Схема алгоритма включения в сбалансированное дерево нового элемента:

1. поиск элемента в дереве (неудачный)
2. включение новой вершины и определение результирующего показателя сбалансированности
3. отход по пути поиска с проверкой показателя сбалансированности для каждой проходимой вершины, балансируя в необходимых случаях соответствующие поддеревья

14 билет

14.1 Уровни описания структур данных.

Функциональной спецификацией типа данных называют внешне формальное определение этого типа, не зависящее от языка и машины. Дать формальное определение типа данных - это значит задать множество значений этого типа и множество изображений этих значений вместе с правилом их интерпретации. Функциональная спецификация обеспечивает полное описание данного типа, за исключением внутренней организации объектов.
Например, функциональная спецификация целого типа определяет множество значений этого типа как последовательность цифр со знаком или без знака, полиномиальную интерпретацию любого такого изображения как числа в позиционной системе счисления с основанием 10, операции (+, -, *, деление (div и mod)) с их свойствами (законы ассоциативности, дистрибутивности и т. д.), отношения (=, <>, <, <=, >, >=) и их свойства, а также ряд функций, аргументами и/или результатами которых являются значения этого типа.
Логическое описание - это отображение функциональной спецификации на средства выбранного языка программирования. Если тип существует в выбранном языке, нужно просто описать переменную в соответствии с требованиями языка, например int x. Иначе необходима декомпозиция объекта на такие составные части, которые могут быть описаны как отдельные объекты программы средствами выбранного языка программирования.
Физическое представление - это конкретное отображение на память машины в соответствии с логическим описанием. Такое отображение всегда связано с линеаризацией структуры, так как память машины обычно состоит из некоторых единиц (слов или байтов), пронумерованных последовательными целыми числами, начиная с нуля. Например, запись двумерного массива происходит по столбцам в строку.
Конструктивные особенности памяти как последовательности слов с произвольным доступом обуславливают два вида физического представления:
Сплошное - это представление, при котором объект размещается в памяти машины в непрерывной последовательности единиц хранения. Например, переменная целого типа представляется на физическом уровне одним машинным словом, состоящим из двух, четырех или восьми байтов с последовательными адресами. Часто применяется для таких структур данных, как массив, очередь, стек, дек. Сплошное представление имеет эффективную аппаратную поддержку.
Цепное представление - это такое представление, при котором значение объекта разбивается на отдельные части, которые могут быть расположены в разных участках памяти машины (не обязательно подряд), причем эти участки тем или иным способом связаны «в цепочку» с помощью указателей, то есть они содержат ссылки на следующие части объекта. Цепное представление используется, как правило, для динамических структурных объектов (списки, деревья, очереди, стеки и деки).

14.2 Процедурный тип. Полиморфизмы на си.

Процедурный тип представляет собой класс типов, отдельные типы которого есть множества глобально определенных процедур с идентичной спецификацией, включая и тип результата для функций. Константами процедурных типов являются имена глобальных процедур, также разбитые на непересекающиеся классы равнозначности в соответствии с заголовками этих процедур. Переменные процедурных типов принимают значения из соответствующего спецификации заголовка множества процедур.
Процедурную переменную можно сравнить с однотипной или присвоить ей допустимое значение другой переменной или константы того же типа.
Поскольку для вызова процедуры в конечном счете нужно знать ее адрес, его роль может играть ссылка на процедуру.
Процедурный тип позволяет просто и систематически описывать более универсальные родовые модули с хранимыми процедурами.
В Си и C++ таких тонких понятий, как процедурный тип, просто нет. Вместо этого предусмотрены указатели на функции.

```
typedef double (*binary_function)(double, double); /*Указатель на функцию двух переменных*/
double f(double x, double y)
{
	return x + y;
}
binary_function pf = f;
double z0 = pf(1, 2); // Вызов функции через указатель
double z1 = (*pf)(3, 4); // Тоже вариант вызова
```

15 билет

15.1 Абстракции в языках программирования

Абстракция - это не только отвлечение от чего-то несущественного и потому помогающее лучше отразить суть дела. Это также инструмент познавательной деятельности человека, приводящий к абстрактным понятиям.
АТД - это, по существу, определение некоторого понятия в виде класса (одного или более) объектов с некоторыми свойствами и операциями.
В самой развитой форме в определение АТД входят следующие четыре части:

1. внешность (видимая часть, сопряжение, интерфейс), содержащая имя определяемого типа (понятия), имена операций с указанием типов их аргументов и значений и т. п.
2. абстрактное описание операций и объектов, с которыми они работают
3. конкретное (логическое) описание этих операций на обычном распространенном языке программирования предыдущего поколения
4. описание связи между 2 и 3, объясняющее, в каком смысле часть 3 корректно представляет часть 2

Простейший пример абстракции в программировании - понятие переменной, сопоставляющей какой-то последовательности ячеек в памяти имя, по которому к ней можно обратиться, и в случае типизированного языка - интерпретации этой последовательности.

Другая значимая абстракция - это абстракция кода, известная как функциональная (процедурная) абстракция: вместо того, чтобы каждый раз писать объёмный код выполнения какой-то операции каждый раз, когда это требуется, описывается функция, эти действия инкапсулирующая. Это очень удобно: во-первых, композиция сложных последовательностей действий сильно запутывает код, а во-вторых - любое изменение придётся многократно дублируя, при этом почти гарантированно создавая ошибки.

15.2 Сортировка Шелла

Д . Шеллом было предложено усовершенствование сортировки вставкой. Для ускорения он предложил выделять в сортируемой последовательности периодические подпоследовательности регулярного шага, в каждой из которых отдельно выполняется обычная (или двоичная) сортировка вставкой.

Эти подпоследовательности пронизыва- ют крупными стежками всю сортируемую последовательность и, по построению, дают большие перемещения элементов. Ввиду регулярного шага (периода) и прямого доступа к элементам массива (индексированного, конечно же, целым типом) прохождение этих подпоследовательностей реализуется циклом с параметром for.

После каждого прохода шаг подпоследовательностей уменьшается и сортировка повторяется с новыми прыжками, причем переход к меньшему шагу не только не требует затрат на организацию новых подпоследовательностей, но и не ухудшает сортированность упорядочиваемого массива

Из интернета:

При сортировке Шелла сначала сравниваются и сортируются между собой значения, стоящие один от другого на некотором расстоянии d.

После этого процедура повторяется для некоторых меньших значений d, а завершается сортировка Шелла упорядочиванием элементов при d=1 (то есть обычной сортировкой вставками).

Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места (в простых методах сортировки, например, пузырьковой, каждая перестановка двух элементов уменьшает количество инверсий в списке максимум на 1, а при сортировке Шелла это число может быть больше).

Невзирая на то, что сортировка Шелла во многих случаях медленнее, чем быстрая сортировка, она имеет ряд преимуществ:

- отсутствие потребности в памяти под стек;
- отсутствие деградации при неудачных наборах данных — быстрая сортировка легко деградирует до $O(n^2)$, что хуже, чем худшее гарантированное время для сортировки Шелла

Элемент d выбирается чаще всего как число N/2, где N- число сортируемых элементов. c

Код из интернета:

```
void ShellSort(int n, int mass[])
{
    int i, j, step;
    int tmp;
    for (step = n / 2; step > 0; step /= 2)
        for (i = step; i < n; i++)
        {
            tmp = mass[i];
            for (j = i; j >= step; j -= step)
            {
                if (tmp < mass[j - step])
                    mass[j] = mass[j - step];
                else
                    break;
            }
            mass[j] = tmp;
        }
}
```

Выигрыш Шелла в том, что на каждом шаге либо сортируется мало элементов, либо они досортировываются.

Недостатки – неустойчивость и сложный математический анализ (до сих пор не найдена универсальная последовательность шагов, дающая минимальную сложность). Сложность **O (n ^ (1+б)), где б от “0” до “1”, что хуже, чем n * ln n**

16 Билет

16.1 Абстрактные типы данных. Пример модуля АТД ОЧЕРЕДЬ.

(определения можно взять из 15.1)

АТД - это определение некоторого понятия в виде класса объектов с некоторыми свойствами и операциями. В сущности, АТД - это прообраз классов в ОО-языках. Как правило, у АТД имеется интерфейс, предназначенный для клиентского кода, и обобщённое описание, достаточное для понимания свойств и задания АТД.

Рассмотрим для примера АТД очередь, подходящую для описания любого явления, удовлетворяющего описанию "первый пришёл - первым ушёл": это могут быть процессы, ожидающие своей доли процессорного времени, люди, стоящие в банке, или даже пары газа в прямой цилиндрической турбине.

```
#ifndef __functions_h_

#define __functions_h_

#include <stdio.h>

typedef struct elem elem;

struct elem{

int val;

elem *next;

};

typedef struct{

elem *top;

elem *bottom;

} stack;

void stack_init(stack *s); // создание

bool empty(stack *s); // “пусто?”

voidpop(stack *s); // удалить первый элемент

inttop(stack *s); // значение первого элемента

voidpush(stack *s, intval); // добавить элемент в конец

voidprint_list(stack *s); // распечатать содержимое

intcount(stack *s); // “сколько элементов?”

#endif

#include "functions.h"
```

```
void stack_init(stack *s){

s->top = 0; }

bool empty(stack *s){

return s->top == 0;

}
```

16.2 Обменные сортировки (пузырьковая сортировка)

В данном разделе мы опишем простой метод сортировки, в котором обмен местами двух элементов производится не с целью вставки или выборки, а непосредственно для упорядочения. Так называемый алгоритм прямого обмена, основывается на сравнении и перестановке пар соседних элементов до тех пор, пока таким образом не будут упорядочены все элементы. Часто эту сортировку называют пузырьковой.

```
void BubbleSort(int a[])
{
    int i, j, k, x;
    for(i = 1; i < n; i++)
        for(j = n-1; j > i; j--)
        {
            if(a[j-1] > a[j])
                  {
                      x = a[j-1];
                      a[j-1] = a[j];
                      a[j] = x;
                  }
        }
}
```

Если на каждом проходе вести подсчет числа состоявшихся обменов (или фиксировать сам факт обменов), то можно улучшить пузырьковую сортировку. Еще одно улучшение - не только фиксировать наличие обмена, но и запоминать индекс последнего состоявшегося обмена k. Поскольку все пары соседних элементов выше индекса k уже упорядочены, просмотры можно заканчивать на этом индексе, большем нижнего предела параметра цикла i.

Если чередовать направления прохода, то сортировка станет Шейкерной.

Число сравнений: $C = n2-n2$

Максимальное число перестановок: $M = 3(n2-n)2$