**Лекция № 4.** Тьюринг әмбебап программасы. Жиындар нумерациясы, алгоритмдік қасиеттері.

**МАҚСАТЫ:** Тьюринг әмбебап программасы, жиындар нумерациясы, алгоритмдік қасиеттері ұғымдарымен таныстыру.

**СҰРАҚТАР**:

1. Тьюринг әмбебап программасы.
2. Жиындар нумерациясы, алгоритмдік қасиеттері.

**НЕГІЗГІ ҰҒЫМДАР:** Тьюринг әмбебап программасы, жиындар нумерациясы, алгоритмдік қасиеттері

**Әмбебап Тьюринг программасы**

Кез келген *A* = {*a*1, *a*2 ,..., *an*} берілген алфавитімен жұмыс жасайтын бағдарламаның жұмысын боямалайтын U универсалды тьюринг программасын қарастырайық. Бұл U программасы, кірісте белгілі бір түрде кездейсоқ Т бағдарламасының коды бар және Х псевдосөзді қабылдап, лентада Т бағдарламасының кодын және T(X) псевдосөзін – Т бағдарламасының Х псевдосөзінде жұмысының нәтижесі – қалдыруға тиіс.

 Әмбебапбағдарламаны құрған оқырман өзін компьютерді ойлап таптым деп есептесе болады. Ψ*n* – N ішінен барлық N әрқайсысы 0 нүктесінде анықталып, бағдарламамен есептелінетін, көп дегенде n тьюринг командаларынан тұратын функциялардың жиыны болсын. *s* (*n*) = max *f* (0) функциясын қарастырайық, мұндағы максимум Ψ*n* жиынындағы барлық функциялардан алынады. Әлбетте, *s*(*n*) барлық жерде анықталады және монотонды. Одан бөлек әділ.

 **Теорема.** Кез келген барлық жерде анықталған есептелетін функция: *f* : *N* → *N* үшін, кез келген *m* ≥ *k* жағдайында *f* (*m*) < *s* (*m*) теңсіздігі орындалатын *k* ∈ *N* бар болады.

Расында, *f* (*n*) – кездейсоқ барлық жерде анықталған N-дегі N функциясы болсын, онда, сөзсіз,

*F*(*n*) = max(*f* (3*n*), *f* (3*n* + 1), *f* (3*n* + 2)) + 1

Функциясы да барлық жерде анықталған және есептелетін болады. Унарлы кодта оны k командадан тұратын *TF* бағдарламасы есептесін. Алдымен, лентаға n санын жазатын, содан кейін *TF* секілді жұмыс жасайтын *T* = [[1,*r*]*n*, *TF*] бағдарламасын қарастырайық. Оның 2n+k командадан тұратыны және кейбір *F* ′ ∈ Ψ2*n+k* функциясын есептейтіні анық.

*F* ′ және s Анықтамасы бойынша, *F*(*n*) = *F*′(0) ≤ *s* (2*n* + *k*). s функциясының монотондылығын қолдана отырып, барлық *n* ≥ *k* үшін *F* (*n*) ≤ *s* (3*n*) теңсіздігін аламыз. Бұдан, *m* ≥ 3*k* болғанда *f* (*m*) < *s* (*m*) орындалады. Дәлелдейтініміз де осы еді.

 ***Салдар.*** *s* (*n*) функциясы есептелмейді, себебі ол ол кез келген есептелінетін функциядан жылдам өседі.

Кез келген белгілі n де s(n) мәнін барлық ұзындығы n-нан кіші бағдарламаларды таңдау арқылы және олардың әрқайсысы үшін тоқтағанға дейінгі уақытта есептеуге болатынын айта кетелік немесе оның шешілметіндігінің дәлелі. Бірақ бағдарламаның шешілмейтілдігі жөніндегі сұрақ жалпы алгоритмді түрде рұқсат етілмейді. Осы тұжырымның басты кезеңдерін анықтап кетелік.

 Т – А алфавитінде жұмыс жасайтын Тьюринг бағдарламасы болсын, және kod (*T*) – Т бағдарламасын кодтайтын А алфавитіндегі сөз. (толық қарастырмаймыз)

 Егер кірісіне өз кодын енгізгенде, ол ақырлы қадамдар санынан кейн тоқтаса, Т бағдарламасы өзді қолданбалы деп аталады, кері жағдайда бағдарлама өзді қолданбалы емес. М – өзді қолданбалы бағдарламалардың кодтарының жиыны болсын.

**Жиындар нумерациясы, алгоритмдік қасиеттері**

1-шi анықтама. Егер Z0 терiс емес бүтiн сандардың жиыны мен арасында өзара бiрмәндi бейне орнатуға болса және көп болса X жиыны саналатын атайды.

2-шi анықтама. Егер жиын саналатын және ақырлы болса, онда санаудан аспайтын деп атайды.

3-шi анықтама. X жиынының санамалауы немесе нөмірлеуі (нумерация) деп Z0 жиынын X жиынына  бейнелеуді айтады.
Егер, f бейнелеуі бірмәнді болса, онда f бейнелеуін санамалау немесе қайталаусыз нөмірлеу деп айтады.

***Теорема.*** М жиыны алгоритмді шешімсіз.

***Дәлелденуі.*** М – алгоритмді шешімді болсын. Онда *T*1 тек М жиынындағы сөздерге тоқтайтын, ал *T*2 – тек *A\** \ *M* жиынындағы сөздерге тоқтайтын *T*1 және *T*2 бағдарламалары бар болады.

 Онда, егер *T*2 өз кодында тоқтаса, kod (*T*2) ∈ ∈ M. Алайда *T*2 анықтамасы бойынша kod (*T*2) ∉ M. Егер де *T*2 өз кодында тоқтамаса , онда М анықтамасы бойынша kod(T2) ∉ M, ал T2 анықтамасы бойынша kod(T2) ∈ M. Сонымен қалайда қарама-қайшылыққа жолығамыз.

Алгоритмді шешілмейтін жиынның тағы бір мысалы М1 бос кірісте тоқтайтын бағдаламалар жиыны болып табылады. Егер М1 шешімді болса, М жиыны да шешімді болатынын көрсету оңай.



Бүтінсанды тор нүктелерінің нөмірленуі, бұл  және жиындары арасында  бірмәнді бейнеленуін жасайды

Есептің алгоритмдік қиындығын есептеу жайында

Тәжірибе жүзінде көптеген қызықты есептерді есептеу автоматтарының көмегімен шешу кезінде сәйкес алгоритмдермен шешуге қажетті жұмыс уақыты мен жады көлемі маңызды роль атқарады.Айтылып кеткен мінездемелер сәйкесінше, уақыттық және кеңістіктік алгоритм қиындығы деп аталады.

**Алгоритмдер (немесе автоматтар) класындағы уақыттық немесе кеңістіктік тапсырма қиындығы** дегеніміз қарастырылған тапсырманы шешуге арналған алгоритмге (автоматқа) берілген класста жадының көлемі мен уақыты.

Күрделі сипаттамалық есептерді табу туралы сұрақ барынша қиын. Қиындықтар қарастырылып отырған есептердің логикалық және кез-келген әмбебеп алгоритмдердегі қиындықтарымен байланысты.

Тьюрингтік бағдарламалар мысалында кеңінен қарастырайық. Есеп кейбір *A* алфавитінде жататын және берілген *u* ∈ *A\** сөз бойынша *v* ∈ *A\** сөзі *R* (*u*, *v*) ақақат болатындай екіорынды сөздік предикатпен *R* (*u*, *v*) ұсынылған деп есептейік.

*v* сөзін *u* кіріс сөзі болғандағы *R* есебінің шешімі деп есептейміз.Бос орындар бойынша *v* сөзі шешімнің болмауы деп қабылдау үшін *R* ге келесі шектеулерді қоямыз:

*R* (λ, λ),

∀*u* ∃*v R* (*u*, *v*),

∀*u* [*R* (*u*, λ) → ∀*v* [*v* ≠ λ → ¬*R* (*u*, *v*)]].

Т тьюрингтік бағдарламаның қорытындысы u кіріс сөзінде *T* (*u*) деп белгілейтін *T* (*u*) шығыс сөзіне тең болып табылады.

Егер кез-келген u кіріс сөзінде ол соңғы қадам саны арқылы тоқтайтын және ∀*u R* (*u*, *T* (*u*)) болса, онда тьюрингтік бағдарлама *T* тапсырманы *R* шешеді.

 time (*T*, *u*) арқылы  *u* кіріс сөзі арқылы жұмыс жасағанда *T* бағдарламасы арқылы бастапқы моменттен тоқтау моментіне дейінгі элементар тьюринг командаларын белгілейміз. Егер *u* кіріс сөзі кезінде *T* бағдарламасы шексіз қадамдар санын орындайтын болса, онда time (*T*, *u*) = ∞ деп есептейміз.

 time (*T*, *u*) шамасын *u* кіріс сөзі кеіндегі *T* бағдарламасының жұмыс уақыты деп атаймыз. Көп жағдайда бұл шама *u* кіріс сөзінің ұзындығына тәуелді, сондықтан *t* (*T*, *n*) = = max time қызығушылық тудырады, мұнда максимум *n* ұзындықты барлық сөздер бойынша есептеледі.

Бұл жағдай жалпы емес екенін байқаймыз; кейбір жағдайларда time (*T*, *u*) шамасы *u*  сөзінің ұзындығына тәуелді болмайды. Мысалы, екілік кодтағы санның дұрыстығын анықтау керек болсын. Ол үшін оның кішкентай разрядына қарау жеткілікті.

*R тапсырмасының уақыттық күрделілігі деп f* (*n*) = min *t* (*T*, *n*) функциясын атауға болады, мұндағы минимум *R* тапсырмасын шешетін барлық Т бағдарламалар арқылы есептеледі. Бірақ мұндай функцияның болуы шарт емес. Мұндай функциялардың болуы есептеуіш алгоритмдерінің класстарына шектеулер қойғанда ғана мүмкін болады. Тәжірибеде нақты алгоритмдердің жұмыс уақыты үшін жоғарғы және төменгі бағалау функцияларын табумен шектеледі.

Төменгі тривиалды емес бағалауларды алу қиын математикалық есептен тұратынын байқаймыз. Мысалы, тьюрингтік бағдарламалар арқылы сөздердің симметриясын түсінудің уақыттық қиындығы төменгі жағында *cn*2 функциясымен бағаланады, мұндағы *n* – сөздің ұзындығы, ал *c* – тұрақты.

Көптеген практикалық көзқарастан қарағанда белгілі жоғарғы бағалаулар есептеуіш құрылғыларды сәйкес алгоритмдерді орындау кезінде үлкен шығындар туралы куәландыратын экспоненциалды сипатта болады.

Егер *R* тапсырмасы үшін n жоғарғы бағалау функциясынан тұратын полиномиал бар болса, онда *R* полиномиалды уақытта рұқсат етілген деп атайды.

Көптеген есептер үшін жоғарғы полиномиалды бағалау функцияларын орнату мүмкін болмайды. Соған қарамастан, *u*, *v*, жұптарында түсіну *R* тапсырмасының шешімі болып табылатын кірісінде *u* болатын *v* сөзі болады *u*, ұзындығынан кейбір полиномдармен шектелген әрбір u үшін жауап *v* болатын *u* сөзінің ұзындығынан полиномиалды шешіледі. Бұл полиномиалды уақытта жауабымен тексерілетін тапсырмалар деп аталады.

Мұндай тапсырма келесі тұжырымнан тұратын бульдік формулалардың орындалуы туралы тапсырма деп аталады. Берілген бульдік фомула арқылы бульдік функция 1 мәнін қабылдаған кездегі айнымалылардың мәнінің жиынын табу. Минималды бағдарламалау тәжірибе бар болса полиномиалды уақыт кезінде осы тапсырманың жауаптарын тексеру мүмкіндігіне көз жеткізуге болады.

Полиномиалды уақыт кезінде тексерілген есептер экспоненциалды көрсеткіш болып табылады. *R* (*u*, *v*) тапсырма болсын, және мүмкін жауаптың ұзындығы *v* кіріс ұзындығынан *u* көпмүше *p* арқылы шектелген, яғги | *v* | ≤ *p* (| *u* |). Әрі қарай *q* – полином, жауаптарды тексеретін бағдарламаның жұмыс уақытының жоғарғы бағасы болып табылады, барлық мүмкін 2 *p* (| *u* |)  *p* (| *u* |) ұзындықты сөздерді және әр қайсысын тексеруге *q* (| *u* |) уақыт жоғалту арқылы жоғарғы *p*(|*u*|) бар алгоритм аламыз.

 *R* полиномиально тапсырмасы *R' тапсырмасына келтіріледі*, егер *Т*1 және *Т*2 полиномиальді уақытта жұмыс жасайтын тьюрингті бағдарламалар бар болса:

∀*u* ∀*v'* [*R'* (*T*1 (*u*), *v'* ) → *R* (*u*, *T*2 (*v'* ))].

Бұл анқытамадан кірісінде *u* бар *R* тапсырмасын шешу үшін келесілер жеткілікті екені шығады

• *u'* = *T*1 (*u*) есептеу,

• содан соң кірісінде u' бар R' тапсырманың v' жауабын табу

• содан соң *v'ке T*2 бағдарламасын қолдану, кірісінде *u* болатын *R* тапсырмасының *v* = *T*2 (*v'* ) жауабын аламыз.

Бұл жағдайда, *v* жауабын алудың келесі сызбасын көрсетеміз:

u → u' → v' → v.

Полиномиальді тексеретін жауабы бар есептердің ішінде кез-келген жауап полиномиальді тексеретін жауап бар екен. Мұндай есептер әмбебап жинақталған есептер деген атқа ие болды.

Тарихта әмбебап жинақталған есептерді 1971 жылы американ математигі Кук ашқан, ол бульдік формуланың орындалу есебі әмбебап жинақталған есеп болатындығын дәлелдеген.

Сол кезде көптеген белгілі есептердің *q* (*u*)·2 бағасы бар әмбебап жинақталған есеп екендігі дәлелденген.

Қазіргі кезде мұндай есептер шеңбері кеңейіп келеді.

**Күрделіліктің негізгі бағалаулары:**

Алгоритмнің күрделілігінің f(n) функциясының негізгі бағалауы **Θ бағасы**. Мұндағы n − деректер көлемінің шамасы немесе кірістің ұзындығын білдіреді.

Алгоритмнің күрделілігінің негізгі бағасы f(n)= **Θ** (g(n))

егер g > 0 , n > 0 болғанда мынандай оң с1, с2, n0, болса : с1 g(n) <= f(n) <= с2 g(n)

n > n0 болғандамынандай с1 мен c2 табуға болады, үлкен n үшін f(n) мәні мына с1 g(n) және с2 g(n) екеуінің аралығында болады.

Бұл жағдайда g(n) функциясын f(n) функциясының асимтотикалық нақты бағасы деп атайды.

Мысалы heapsort сұрыптау әдісі үшін күрделілік бағасы f(n)= **Θ** (*nlogn*) болады, яғни *g*(*n*) = *n*log*n.*

f(n)= **Θ** (g(n)) теңдіктен *g*(*n*) = **Θ** (f(n)) туындайды.

Маңыздысы: **Θ** (g(n)) функция емес, функциялардың жиыны, ол f(n)-нің өсуін тұрақты көбейткішке дейін дәлдікпен көрсетеді.

**Θ *бағасы*** бір мезгілде функция өсімінің жоғары және төменгі бағаны береді. Көбінесе бұл бағаларды жеке қарастыруға тура келеді.

***О бағасы*** алгоритм қиындығының, яғни f(n) функциясының өсуінің жоғарғы асимптотикалық бағасын береді. f(n)= ***О***(g(n)) болады, егер

 орындалса.

Басқаша айтқанда тұрақты көбейткішке дейін дәлдікпен g(n) функциясына қарағанда f(n) функциясы жылдам өспейтін функциялар класын анықтайды.

**Ω *бағасы*** алгоритм қиындығының, яғни f(n) функциясының өсуінің төменгі асимптотикалық бағасын береді және тұрақты көбейткішке дейін дәлдікпен g(n) функциясына қарағанда f(n) функциясы баяу өспейтін функциялар класын анықтайды.

f(n)= **Ω** (g(n)) болады, егер



Егер f(n)= ***О***(g(n)) және f(n)= **Ω** (g(n)) орындалса ғана, f(n)= **Θ** (g(n)) теңдігі орындалады

Алгоритмдерді асимптотикалық талдауды практикалық та , теориялық та маңызы зор. Мысалы,элементтерді жұптап сұрыптауға негізделген барлық сұрыптау алгоритмдері n элементті **Ω**(*nlogn*) кем емес уақытта сұрыптайды.

**ҚОРЫТЫНДЫ:**

 Търингтің әмбебап машинасы деп Търингтің кез келген машинасын алмастыра алатын машинаны айтады.

 Егер Z0 терiс емес бүтiн сандардың жиыны мен арасында өзара бiрмәндi

X жиынының санамалауы немесе нөмірлеуі (нумерация) деп Z0 жиынын X жиынына  бейнелеуді айтады. Егер, f бейнелеуі бірмәнді болса, онда f бейнелеуін санамалау немесе қайталаусыз нөмірлеу деп айтады.