**№5-LABORATORIYA ISHI**

* 1. **Rekursiya va ularni dasturlashda ishlatish.**
  2. **Rekursiv va iterativ algoritmlarni ishlatishga misol.**

**Ishdan maqsad: Ishdan maqsad**: Rekursiya va ularni dasturlashda ishlatishga oid amaliy ko‘nikmaga ega bo‘lish va berilgan topshiriqlar bo‘yicha dastur tuzish, natijani tahrirlash.

**Qo‘yilgan masala:** Topshiriq variantidagi masalani so‘ralayotgan rekursiya usuli yordamida yechishning Python tilidagi dasturini yaratish ko‘nikmasiga ega bo‘lish.

***Ish bajarish tartibi:***

* Tajriba ishi nazariy ma’lumotlarini o‘rganish;
* Berilgan topshiriqning algoritmini ishlab chiqish;
* Python dasturlash muhitida dasturni yaratish;
* Natijalarni tekshirish;
* Hisobotni tayyorlash va topshirish.

**5.1. Rekursiya va ularni dasturlashda ishlatish.**

Agar 1 dan n gacha bo'lgan sonlar yig'indisini bilmoqchi bo'lsam, bu erda n natural son bo'lsa, men qo'lda *1 + 2 + 3 + 4 + ... + …+ n* ni hisoblashim mumkin edi. Yoki siz shunchaki **loop** yozishingiz mumkin for:

*n = 0*

*for i in range (1, n+1):*

*n += i*

Yoki rekursiyadan foydalaning:

*def rekursiya(n):*

*if n == 1:*

*return 1*

*return n + rekursiya(n - 1)*

Rekursiya birinchi ikki usulga nisbatan bir qancha afzalliklarga ega. Rekursiya 1 dan 3 gacha bo'lgan sonlar uchun *1 + 2 + 3* ni yozishdan ko'ra kamroq vaqt oladi. *Rekursiya (4*) uchun rekursiya teskari yo'nalishda ishlashi mumkin:

Funksiya chaqiruvi: *(4 -> 4 + 3 -> 4 + 3 + 2 -> 4 + 3 + 2 + 1 -> 10)*

**for** sikli qat'iy oldinga ishlaydi: *(1 -> 1 + 2 -> 1 + 2 + 3 -> 1 + 2 + 3 + 4 -> 10).* Ba'zan rekursiv yechim iterativ yechimga qaraganda oddiyroq bo'ladi. Bu bog'langan ro'yxatni o'zgartirishni amalga oshirishda aniq.

*Rekursiv funksiya* - bu o'zini chaqiradigan funksiyadir.

Eng oddiy misol uchun quyidagi kodni ko'rib chiqamiz:

*def fac\_rek(n):*

*if n == 1 or n==0:*

*return n*

*else:*

*return n\*fac\_rek(n-1)*

Bu yerda rekursiv funksiyani chaqirish va unga butun sonni berish orqali siz ushbu raqamning faktorialini olasiz ***(n!).***

***Faktoriallar haqida qisqacha.*** *Sonning faktoriali* - bu har bir oldingi songa 1 ga ko'paytiriladigan son.

***Masalan***, 7!:

*7 ! = 7 \* 6 \* 5 \* 4 \* 3 \* 2 \* 1 = 5040*

Funksiyadan foydalanib, raqamning faktorialini hisoblashimiz mumkin:

*num = 3*

*print(f"Factorial {num} bu {factorial\_recursive(num)}")*

Bu funksiya chiqadi: "3 faktorial 6". Ushbu rekursiv funksiyani yana bir bor ko'rib chiqing:

*def factorial\_recursive(n):*

*...*

Muntazam funksiyaga o'xshab, rekursivning nomi dan keyin ***def***, parametr esa qavs ichida ko'rsatilgan ***n***:

*def factorial\_recursive(n):*

*if n == 1:*

*return n*

*else:*

*return n\*factorial\_recursive(n-1)*

Shartli konstruksiya tufayli o'zgaruvchining nqiymati 1 ga teng bo'lsagina qaytib keladi. Buni tugatish sharti ham deyiladi. Shartlar bajarilganda rekursiya to'xtaydi.

*def factorial\_recursive(n):*

*if n == 1:*

*return n*

*else:*

*return n\*factorial\_recursive(n-1)*

Yuqoridagi kodda rekursiyaning bir qismi ajratilgan. ***else*** shartli qurilish blokida mahsulot ***n*** va parametr bilan bir xil funksiyaning qiymatlari qaytariladi ***n-1***.

### Rekursiv funksiya qanday ishlashi haqida tafsilotlar

Rekursiya qanday ishlashini yaxshiroq tushunish uchun 3-parametrli funksiyani bajarish jarayonini bosqichlarga ajratamiz.

Buning uchun quyida biz har bir misolni haqiqiy raqamlar bilan taqdim etamiz. Bu argument sifatida 3 qiymatiga ega bitta funksiyani chaqirganingizda nima sodir bo'lishini "kuzatish"ga yordam beradi:

*# Birinchi chaqiruv*

*factorial\_recursive(3):*

*if 3 == 1:*

*return 3*

*else:*

*return 3\*factorial\_recursive(3-1)*

*# Ikkinchi chaqiruv*

*factorial\_recursive(2):*

*if 2 == 1:*

*return 2*

*else:*

*return 2\*factorial\_recursive(2-1)*

*# Uchinchi chaqiruv*

*factorial\_recursive(1):*

*if 1 == 1:*

*return 1*

*else:*

*return 1\*factorial\_recursive(1-1)*

Rekursiv funksiya ifoda uchun javobni bilmaydi *3\*factorial\_recursive(3–1)*, shuning uchun u stekga boshqa chqiruvni qo'shadi.

## ***Rekursiya qanday ishlaydi?***

*/ \ factorial\_recursive (1) - oxirgi chaqiruv*

*|| factorial\_recursive (2) - ikkinchi chaqiruv*

*|| factorial\_recursive (3) - birinchi chaqiruv*

Yuqorida stek qanday yaratilganligi ko'rsatilgan. Bu *LIFO* *(oxirgi kiruvchi, birinchi chiqadi)* jarayoni bilan bog'liq. Esda tutingki, funksiyaga birinchi chqiruvlar javobni bilmaydi, shuning uchun ular stekga qo’shiladi.

Ammo stekga chqiruv qo'shilishi bilanoq, *factorial\_recursive(1)* javob bor, stek barcha hisob-kitoblarni haqiqiy qiymatlar bilan amalga oshirib, teskari tartibda "ochila" boshlaydi. Ushbu jarayonda qatlamlarning har biri alohida jarayonda tushadi.

* *factorial\_recursive (1) tugallaydi, 1 ni yuboradi*
* *factorial\_recursive (2) va stekdan chiqib ketdi.*
* *factorial\_recursive (2) yakunlaydi, 2 \* 1 ga yuboradi*
* *factorial\_recursive (3) va stekdan chiqib ketdi. Nihoyat, bu erda else iborasi tugaydi, 3 \* 2 = 6 qaytariladi va oxirgi qatlam stekdan chiqariladi.*

***Pythonda rekursiya 3000 qatlam bilan cheklangan.***

*>>> import sys*

*>>> sys.getrecursionlimit()*

*3000*

Shuningdek, siz "parallel" rekursiv funksiya chaqiruvlariga ega bo'lishingiz mumkin. Misol uchun, Fibonachchi ketma-ketligini ko'rib chiqamiz:

*Agar raqam 0 bo'lsa, javob 0 bo'ladi.*

*Agar raqam 1 bo'lsa, javob 1 bo'ladi.*

*Aks holda, javob oldingi ikkita Fibonachchi sonining yig'indisidir.*

Biz buni quyidagicha belgilashimiz mumkin:

*def fib(n):*

*if n == 0 or n == 1:*

*return n*

*else:*

*return fib(n - 2) + fib(n - 1)*

Bu funksiyani bilan bo'lgani kabi batafsil ko'rib chiqmayman *factorial(3)*, lekin yakuniy qaytarish qiymati *fib(5)* quyidagi (sintaktik jihatdan yaroqsiz) ifodaga teng:

*(*

*fib((n - 2) - 2)*

*+*

*(*

*fib(((n - 2) - 1) - 2)*

*+*

*fib(((n - 2) - 1) - 1)*

*)*

*)*

*+*

*(*

*(*

*fib(((n - 1) - 2) - 2)*

*+*

*fib(((n - 1) - 2) - 1)*

*)*

*+*

*(*

*fib(((n - 1) - 1) - 2)*

*+*

*(*

*fib((((n - 1) - 1) - 1) - 2)*

*+*

*fib((((n - 1) - 1) - 1) - 1)*

*)*

*)*

*)*

*Yechim (1 + (0 + 1)) + ((0 + 1) + (1 + (0 + 1))) bo'ladi 5.*

***Endi yana bir nechta atamalarni ko'rib chiqamiz:***

Rekursiyaning dumi oddiygina rekursiv funksiya chaqiruvidir, bu oxirgi operatsiya bo'lib, qiymatni qaytarishdan oldin bajarilishi kerak. Aniq bo'lish uchun, return *foo(n - 1)*bu rekursiyaning dumi, lekin qaytish *foo(n - 1) +* *1* emas (chunki qo'shish operatsiyasi oxirgi operatsiya bo'ladi).

*Tail of Call Optimization (TCO)* rekursiv funksiyalarda rekursiyani avtomatik ravishda kamaytirish usulidir.

*Tail Call Elimination (TCE*) - bu rekursiyasiz baholanishi mumkin bo'lgan iboraga chaqiruv qilishning qisqartmasi. *TCE - TCO* ning bir turi.

Chaqiruvlarni optimallashtirish bir necha sabablarga ko'ra foydali bo'lishi mumkin:

Rekursiya muhitlar tomonidan ishlatiladigan xotira hajmini kamaytirishi mumkin. Hech kim cheksiz xotiraga ega emasligi sababli, ortiqcha rekursiv funksiya chaqiruvlari stekning to'lib ketishiga olib keladi.

Tarjimon stek ramka kalitlari sonini kamaytirishi mumkin.

Python bir necha sabablarga ko'ra TCOga ega emas, shuning uchun bu cheklovni hal qilish uchun boshqa usullardan foydalanish mumkin. Amaldagi usul foydalanish holatiga bog'liq. Intuitiv ravishda, factorial yoki fibonachi nisbatan osonlik bilan quyidagi kabi iterativ kodga aylantirilishi mumkin:

*def factorial(n):*

*product = 1*

*while n > 1:*

*product \*= n*

*n -= 1*

*return product*

﻿

*def fib(n):*

*a, b = 0, 1*

*while n > 0:*

*a, b = b, a + b*

*n -= 1*

*return a*

Bu odatda qo'lda rekursiyani yo'q qilishning eng samarali usulidir, ammo murakkabroq funksiya lar uchun bu qiyin bo'lishi mumkin.

Endi siz Pythonda rekursiyadan qanday qochish kerakligini bilasiz, lekin uni qachon ishlatishingiz kerak? Javob "tez-tez emas". Barcha rekursiv funksiyalar iterativ tarzda amalga oshirilishi mumkin. Buni qanday qilish kerakligi haqida gap boradi. Biroq, rekursiyadan foydalanish mumkin bo'lgan kamdan-kam holatlar mavjud. Rekursiya Pythonda keng tarqalgan bo'lib, bu erda kutilgan kirishlar sezilarli miqdordagi rekursiv funksiya chaqiruvlariga olib kelmaydi.

Yuqoridagi Fibonachi ketma-ketligi misolida ta'rifni Pythonda qanday qo'llash va keyinchalik *lru* keshidan foydalanish yaxshi ko'rsatilgan bo'lsada, u 2 ta rekursiv chaqiruvlarni amalga oshiradiganligi sababli samarasiz ish vaqtiga ega. Funksiya chaqiruvlari soni eksponensial ravishda oshadi n.

Bunday holda, chiziqli rekursiyadan foydalanish yaxshidir:

*def fib(n):*

*if n <= 1:*

*return (n,0)*

*else:*

*(a, b) = fib(n - 1)*

*return (a + b, a)*

Ammo bu misolda bir juft raqamni qaytarishda muammo bor. Bu shuni ko'rsatadiki, rekursiyadan foydalanish har doim ham foydali emas.

**5.2. Rekursiv va iterativ algoritmlarni ishlatishga misol**

# ***Iterativ va rekursiv algoritmlar.*** Umumiy holda, iteratsiya ma'lumotlarga ishlov berishni tashkil etishning ma'lum bir ketma-ketligi ko'p marta takrorlanadigan tarzda tushuniladi.

Takrorlashga asoslangan algoritm *iterativ* *(iterativ)* deb ataladi. Siklik konstruktsiyalar takrorlanuvchi algoritmlarga misol bo'la oladi.

Takrorlash kontseptsiyasining umumlashtirilishi rekursiya bo'lib, u algoritm o'zidan yordamchi algoritm sifatida foydalanadigan ma'lumotlarni qayta ishlashni tashkil qilish usuli sifatida tushuniladi. Rekursiv algoritmlarning klassik misollari faktorial va Fibonachchi raqamlarini hisoblashdir.

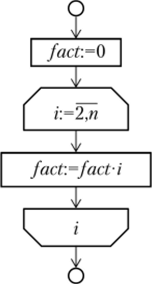
*n* butun sonning *faktoriali* rekursiv ravishda quyidagicha aniqlanadi



Ba'zi *n>* sonning faktorialini hisoblash uchun *(n -* 1) sonning faktorialini hisoblash kerak, buning uchun esa, o'z navbatida, sonning faktorialini *(n -* 2) va hokazolarni hisoblash kerak. .

Algoritmni o'ziga chaqirish *to'g'ridan-to'g'ri rekursiya* deyiladi *.*To'g'ridan-to'g'ri chaqiruvlar to'plami ba'zan *rekursiv descent* deb ataladi *.*

To'g'ridan-to'g'ri chaqiruvlarning ma'lum sonidan so'ng, qiymati (1.1) ta'rifi bilan berilgan 1! ni hisoblash kerak bo'ladi. Shundan so'ng, teskari jarayon boshlanadi, bunda oraliq faktorialni hisoblash natijasi ishlatiladi.



*1-sxema. Faktorialni takroriy hisoblash*

Bunday qiymatlarni almashtirish *rekursiya qaytishi* deb ataladi *.*Rekursiv daromadlar to'plami ba'zan *rekursiv ko'tarish* deb ataladi*.*

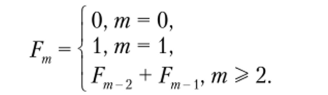
Rekursiv algoritm har doim sikl konstruksiyalari yordamida iterativga aylantirilishi mumkin. Rekursiya muammoning matematik tomonini tabiiyroq aks ettiradigan va/yoki tushunish oson bo'lgan algoritmga olib keladigan hollarda, odatda, iterativ yondashuvdan ko'ra rekursiv yondashuv afzalroqdir.

Faktorial hisoblash algoritmining iterativ analogi shaklda ko'rsatilgan. 1.9.

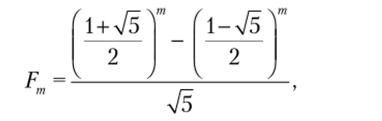
Formula ***(1.1)*** ba'zi bir *n* sonining faktorial qiymati bilan oldingi butun sonning faktorial qiymati o'rtasidagi munosabatni aks ettiradi . Bunday formulalar takrorlanuvchi deb ataladi.

Formula, agar ketma-ketlikning ba'zi *n-* a'zolarining oldingi hadlari bo'yicha bog'liqligini ifodalasa, u takroriy deyiladi .

*Fibonachchi soni F m* rekursiv formula bilan aniqlangan sondir



Birinchi 10 ta Fibonachchi raqamlari ketma-ketligi quyidagicha ko'rinadi: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34. Analitik ifoda ham mavjud (Vine formulasi)



Bu rekursiv formuladan ko'ra og'irroq. Ko'rinib turibdiki, *F m* ning kichik *r* qiymatlari uchun ham muhim ahamiyatga ega, masalan, *F 20 =* 6765, F 30 = 832040, *F i0 =* = 102,334,155 soni *F:m* 63 ta raqamdan iborat (o'nlik kasrlarda).

Faktorial va Fibonachchi raqamlarini aniqlash uchun rekursiv formulalardan foydalanilganda, *aniq rekursiya* qo'llanilishi aytiladi - joriy qiymatni hisoblash uchun faqat original rekursiv formuladan foydalaniladi. In *bilvosita* ( *shubhasiz* ) Takrorlash uchun, recursive formula bilan bir qatorda, boshqa operatsiyalar, shuningdek, ishlatiladi.

Bilvosita rekursiyali algoritmga misol sifatida minimal elementni *a v a 2 ,* ..., *a n* raqamlari orasidan qidirishni keltirish mumkin, bunda minimal element kamida ikkita raqamdan aniqlanadi: oxirgi element va minimal. birinchi *(n* - 1) raqamlardan element . Birinchi (r - 1) qiymatlarning minimal elementini topish shunga o'xshash mulohazalar yordamida amalga oshiriladi. Oxir oqibatda, eng kam elementi topish I minimal elementi topish uchun tushgan, va *bir 2 .*

Algoritmlarni kompyuterda amalga oshirishda rekursiyaning mavjudligi biroz xotirani - *chaqiruvlar to'plamini* talab qiladi *.*U nazoratni asosiy protsedura yoki funktsiyaga qaytarish uchun foydalaniladigan bolalar rekursiv chaqiruvlari haqidagi ma'lumotlarni saqlaydi.

Rekursiv algoritmlarni ishlab chiqish jarayoni uch bosqichdan iborat: parametrlash, bazani taqsimlash va parchalanish.

*Parametrlashtirish.*Ushbu bosqichda algoritmning kirish ma'lumotlarini tavsiflovchi parametrlarni tanlash amalga oshiriladi. Misol uchun, hisoblash *, n* hisoblash uchun, bu bir tabiiy raqam / r bo'lgan *F m, u bo'ladi* soni *m.* Raqamlar majmui minimal elementi topish uchun algoritm uchun, raqamlar soni.

*Bazani taqsimlash* natijasi aniq bo'lgan va qo'shimcha operatsiyalarni talab qilmaydigan ahamiyatsiz holatlarning tavsifini o'z ichiga oladi. Rekursiv algoritm oxir-oqibatda aynan shu holatlarga to'g'ri keladi. Faktorialni rekursiv hisoblash uchun asosiy holat 0 ga teng! = 1, Fibonachchi raqamlarini hisoblash uchun - ikkita asosiy holat: *F 0* = 0 va *F {* = 1. Minimal elementni topish algoritmi uchun asosiy holat *min (a v a 2) ni* topish bo'ladi *.*

*Parchalanish.*Bu bosqichda algoritmni bajarishning umumiy holati ko'rib chiqiladi, bu esa ancha "oddiy" holatning bajarilishiga qisqartiriladi. "Oddiylik" deganda biz rekursiv algoritm parametrlari qiymatining pasayishi, ma'lumotlar hajmining pasayishi va boshqalarni anglatishimiz mumkin.

Dekompozitsiya nafaqat umumiy vazifa va kichik vazifalar o'rtasidagi bog'lanishlarni, balki keyingi bosqichda parametrlarning o'zgarishi xarakterini ham tavsiflaydi. Masalan, faktorialni hisoblash uchun parchalanish qoidasi *n* = *n (n* - 1) !, Fibonachchi raqamlarini hisoblashda - *F m = F m \_ 2 + F m \_ v* Minimal elementni topish algoritmi uchun, bunday qoida sifatida yozish mumkin

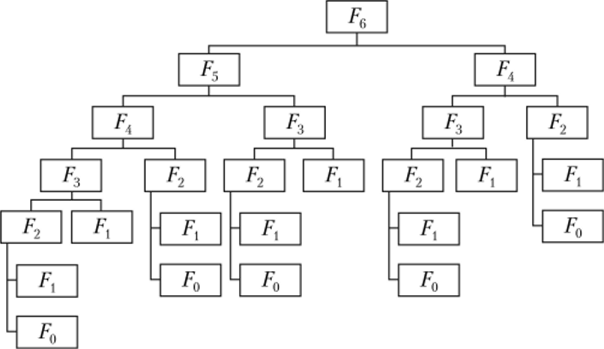


Rekursiyani o'z ichiga olgan algoritmlarning samaradorligini baholashda quyidagi xususiyatlar hisoblab chiqiladi:

* rekursiya chuqurligi – qaytarilmasdan rekursiv algoritm chaqiruvlarining maksimal soni;
* joriy rekursiya darajasi – vaqtning ma’lum bir momentidagi rekursiv chaqiruvlar soni;
* umumiy chaqiruvlar soni – dastur ishlayotgan vaqtda rekursiv *algoritmga* chaqiruvlar soni.

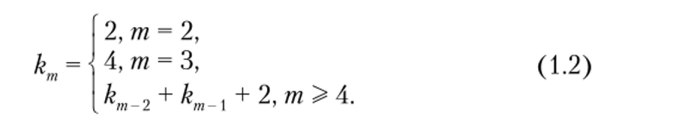
Masalan, *n* ni hisoblashda rekursiya chuqurligi va chaqiruvlarning umumiy soni *n>* joriy rekursiya darajasi doimiy va 1 ga teng.

Fibonachchi raqamlarini hisoblash algoritmining parametrlarini baholash uchun biz chaqiruv daraxti deb ataladigan narsani ko'rib chiqishimiz mumkin (1.10-rasmda F 6 raqamiga misol keltirilgan ).



*2-sxema. Fibonachchi F6 raqamini hisoblash uchun daraxtga chaqiruv qiling*

*F m* ni hisoblashda rekursiya chuqurligi *m n* ekanligini va *Fm* sonini rekursiv hisoblash uchun *k rn* to‘g‘ridan-to‘g‘ri chaqiruvlarning umumiy soni *% /* rekursiv formula bilan aniqlanishini ko‘rish oson.



Formula (1.2) shuni ko'rsatadiki, Fibonachchi raqamlarini rekursiv hisoblash apparat resurslaridan foydalanish nuqtai nazaridan kompyuterni amalga oshirishda juda samarasizdir (chaqiruvlar to'plamining kerakli hajmi). Algoritmning iterativ versiyasi bu borada ancha samarali.

Berilgan misollarda rekursiv algoritmlar iterativ analoglar bilan solishtirganda samarasiz bo'lishiga qaramay, ayrim hollarda ulardan foydalanish dasturni ishlab chiqish jarayonini sezilarli darajada soddalashtirishi mumkin. Bundan tashqari, ulardan foydalanish katta hajmdagi ma'lumotlarni qayta ishlashga sarflanadigan vaqtni sezilarli darajada qisqartiradi.

***1-masala***. Massiv elementlari orasidagi min va max qiymatini topish dasturini tuzing?

***Itratsiya yoramida:***

*def getMin(arr, n):*

*res = arr[0]*

*for i in range(1,n):*

*res = min(res, arr[i])*

*return res*

*def getMax(arr, n):*

*res = arr[0]*

*for i in range(1,n):*

*res = max(res, arr[i])*

*return res*

*arr = [12, 1234, 45, 67, 1]*

*n = len(arr)*

*print ("Massivning minimal elementi: ", getMin(arr, n))*

*print ("Massivning maksimal elementi:", getMax(arr, n))*

***Rekursiya yordamida:***

*def getMin(arr, n):*

*if(n==1):*

*return arr[0]*

*else:*

*return min(getMin(arr[1:], n-1), arr[0])*

*def getMax(arr, n):*

*if(n==1):*

*return arr[0]*

*else:*

*return max(getMax(arr[1:], n-1), arr[0])*

*arr = [12, 1234, 45, 67, 1]*

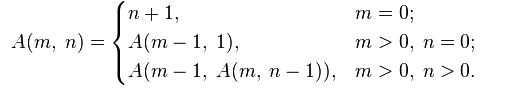
*n = len(arr)*

*print ("Massivning minimal elementi: ", getMin(arr, n))*

*print ("Massivning maksimal elementi:", getMax(arr, n))*

**Topshiriqlar**

* 1. Massiv berilgan, barcha mumkin bo'lgan murakkab sonlarni chop eting? *(Dastur iterative va rekursiv usulda bajarilib qadamlar soni bo’yicha tahlil qiling.)*
  2. Satrdagi unli harflarni sanash dasturini tuzing? *(Dastur iterative va rekursiv usulda bajarilib qadamlar soni bo’yicha tahlil qiling.)*
  3. Satrdagi birinchi bosh harfni aniqlash dasturini tuzing? *(Dastur iterative va rekursiv usulda bajarilib qadamlar soni bo’yicha tahlil qiling.)*
  4. Satrdagi birinchi raqamni aniqlash dasturini tuzing? *(Dastur iterative va rekursiv usulda bajarilib qadamlar soni bo’yicha tahlil qiling.)*
  5. Sonni 10 lik sanoq sistemasidan 2 lik sanoq sistemasiga o’tkazish dasturini tuzing? *(Dastur iterative va rekursiv usulda bajarilib qadamlar soni bo’yicha tahlil qiling.)*
  6. Ro’yxat elementlarini teskari aylantirish dasturini yozing. *(Dastur iterative va rekursiv usulda bajarilib qadamlar soni bo’yicha tahlil qiling.)*
  7. ***Akkerman funksiyasi***. Hisoblash nazariyasida Akkerman A(m,n) funksiyasi muhim rol o'ynaydi, quyidagicha aniqlanadi:



Berilgan m va n ikkita manfiy bo'lmagan butun sonlarni, har biri alohida satrda. A (m, n) qiymatini rekursiv funksiya hosil qilgan holda chop eting.

* 1. Natural sonning n-darajali yig‘indisini topish dasturini yozing. *(Dastur iterative va rekursiv usulda bajarilib qadamlar soni bo’yicha tahlil qiling.)*
  2. N o’lchamli,elementlari musbat butun sonlardan iborat bo’lgan A massivning toq elementlari sonini aniqlovchi dastur tuzing? *(Dastur iterative va rekursiv usulda bajarilib qadamlar soni bo’yicha tahlil qiling.)*
  3. N natural son berilgan, uning ikkining darajasiga teng yoki yo’qligini aniqlovchi dastur tuzing. Agar ushbu son ikkining darajasiga teng bo’lsa “HA”, aks holda “YO’Q” degan qiymat qaytarsin. *(Dastur iterative va rekursiv usulda bajarilib qadamlar soni bo’yicha tahlil qiling.)*